The Art of Error Correcting Coding

Robert H. Morelos-Zaragoza

WILEY

Companion Website

The Art of Error Correcting Coding

The Art of Error Correcting Coding

Robert H. Morelos-Zaragoza
SONY Computer Science Laboratories, Inc. JAPA

JOHN WILEY & SONS, LTD

Copyright © 2002 by John Wiley & Sons, Ltd
Baffins Lane, Chichester,
West Sussex, PO19 1UD, England

National 01243 779777
International (+44) 1243 779777

e-mail (for orders and customer service enquiries): cs-books@wiley.co.uk
Visit our Home Page on http://www.wileyeurope.com or http://www.wiley.com

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except under the terms of the Copyright Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency,
90 Tottenham Court Road, London, W1P 9HE, UK, without the permission in writing of the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the publication.

Neither the author(s) nor John Wiley & Sons, Ltd accept any responsibility or liability for loss or damage occasioned to any person or property through using the material, instructions, methods or ideas contained herein, or acting or refraining from acting as a result of such use. The author(s) and Publisher expressly disclaim all implied warranties, including merchantability of fitness for any particular purpose. There will be no duty on the author(s) or Publisher to correct any errors or defects in the software.

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Ltd is aware of a claim, the product names appear in initial capital or capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Other Wiley Editorial Offices

John Wiley & Sons, Inc., 605 Third Avenue,
New York, NY 10158-0012, USA

WILEY-VCH Verlag GmbH
Pappelallee 3, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 33 Park Road, Milton,
Queensland 4064, Australia

John Wiley & Sons (Canada) Ltd, 22 Worcester Road
Rexdale, Ontario, M9W 1L1, Canada

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01,
Jin Xing Distripark, Singapore 129809

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN 0471 49581 6

Produced from LaTeX files supplied by the author.
Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire.
This book is printed on acid-free paper responsibly manufactured from sustainable forestry, in which at least two trees are planted for each one used for paper production.

Contents

Preface ix

Foreword xi

The ECC web site xiii

1 Introduction 1
1.1 Error correcting coding: Basic concepts 3
1.1.1 Block codes and convolutional codes 3
1.1.2 Hamming distance, Hamming spheres and error correcting capability 4
1.2 Linear block codes 5
1.2.1 Generator and parity-check matrices 6
1.2.2 The weight is the distance 6
1.3 Encoding and decoding of linear block codes 7
1.3.1 Encoding with G and H 7
1.3.2 Standard array decoding 8
1.3.3 Hamming spheres, decoding regions and the standard array 11
1.4 Weight distribution and error performance 12
1.4.1 Weight distribution and undetected error probability over a BSC . . . 12
1.4.2 Performance bounds over BSC, AWGN and fading channels 13
1.5 General structure of a hard-decision decoder of linear codes 19

2 Hamming, Golay and Reed-Muller codes 23
2.1 Hamming codes 23
2.1.1 Encoding and decoding procedures 24
2.2 The binary Golay code 25
2.2.1 Encoding 25
2.2.2 Decoding 26
2.2.3 Arithmetic decoding of the extended (24, 12, 8) Golay code. 26
2.3 Binary Reed-Muller codes 27
2.3.1 Boolean polynomials and RM codes 27
2.3.2 Finite geometries and majority-logic decoding 28

3 Binary cyclic codes and BCH codes 33
3.1 Binary cyclic codes 33
3.1.1 Generator and parity-check polynomials 33

3.1.2 The generator polynomial . . . 34
3.1.3 Encoding and decoding of binary cyclic codes . . . 35
3.1.4 The parity-check polynomial . . . 36
3.1.5 Shortened cyclic codes and CRC codes . . . 37
3.2 General decoding of cyclic codes . . . 39
3.2.1 $GF(2^m)$ arithmetic . . . 41
3.3 Binary BCH codes . . . 44
3.4 Polynomial codes . . . 45
3.5 Decoding of binary BCH codes . . . 46
3.5.1 General decoding algorithm for BCH codes . . . 48
3.5.2 The Berlekamp-Massey algorithm (BMA) . . . 49
3.5.3 PGZ decoder . . . 52
3.5.4 Euclidean Algorithm (EA) . . . 53
3.5.5 Chien search and error correction . . . 55
3.5.6 Errors-and-erasures decoding . . . 55
3.6 Weight distribution and performance bounds . . . 56
3.6.1 Error performance evaluation . . . 57

4 Non-binary BCH codes: Reed-Solomon codes . . . 63
4.1 RS codes as polynomial codes . . . 63
4.2 From binary BCH to RS codes . . . 63
4.3 Decoding RS codes . . . 64
4.3.1 Remarks on decoding algorithms . . . 69
4.3.2 Errors-and-erasures decoding . . . 69
4.4 Weight distribution . . . 73

5 Binary convolutional codes . . . 75
5.1 Basic structure . . . 75
5.1.1 Recursive systematic convolutional codes . . . 80
5.1.2 Free distance . . . 81
5.2 Connections with block codes . . . 81
5.2.1 Zero-tail construction . . . 81
5.2.2 Direct-truncation construction . . . 82
5.2.3 Tail-biting construction . . . 82
5.2.4 Weight distributions . . . 83
5.3 Weight enumeration and performance bounds . . . 84
5.4 Decoding: Viterbi algorithm with Hamming metrics . . . 86
5.4.1 Maximum likelihood decoding and metrics . . . 87
5.4.2 The Viterbi algorithm . . . 88
5.4.3 Implementation issues . . . 90
5.5 Punctured convolutional codes . . . 96
5.5.1 Implementation issues related to punctured convolutional codes . . . 99
5.5.2 RCPC codes . . . 100

6 Modifying and combining codes . . . 103
6.1 Modifying codes . . . 103
6.1.1 Shortening . . . 103

6.1.2 Extending . . . 105
6.1.3 Puncturing . . . 105
6.1.4 Augmenting and expurgating . . . 106
6.2 Combining codes . . . 108
6.2.1 Time-sharing of codes . . . 108
6.2.2 Direct-sums of codes . . . 109
6.2.3 Products of codes . . . 111
6.2.4 Concatenated codes . . . 117
6.2.5 Generalized concatenated codes . . . 119

7 Soft-decision decoding . . . 123
7.1 Binary transmission over AWGN channels . . . 124
7.2 Viterbi algorithm with Euclidean metric . . . 124
7.3 Decoding binary linear block codes with a trellis . . . 130
7.4 The Chase algorithm . . . 131
7.5 Ordered statistics decoding . . . 133
7.6 Generalized minimum distance decoding . . . 134
7.6.1 Sufficient conditions for optimality . . . 135
7.7 List decoding . . . 136
7.8 Soft-output algorithms . . . 136
7.8.1 Soft-output Viterbi algorithm . . . 136
7.8.2 Maximum-a-posteriori (MAP) algorithm . . . 139
7.8.3 Log-MAP algorithm . . . 141
7.8.4 Max-Log-MAP algorithm . . . 142
7.8.5 Soft-output OSD algorithm . . . 142

8 Iteratively decodable codes . . . 145
8.1 Iterative decoding . . . 147
8.2 Product codes . . . 149
8.2.1 Parallel concatenation: turbo codes . . . 149
8.2.2 Serial concatenation . . . 155
8.2.3 Block product codes . . . 157
8.3 Low-density parity-check codes . . . 161
8.3.1 Tanner graphs . . . 161
8.3.2 Iterative hard-decision decoding: The bit-flip algorithm . . . 163
8.3.3 Iterative probabilistic decoding: belief propagation . . . 164

9 Combining codes and digital modulation . . . 171
9.1 Motivation . . . 171
9.1.1 Examples of signal sets . . . 172
9.1.2 Coded modulation . . . 174
9.1.3 Distance considerations . . . 175
9.2 Trellis-coded modulation (TCM) . . . 176
9.2.1 Set partitioning and trellis mapping . . . 176
9.2.2 Maximum-likelihood decoding . . . 177
9.2.3 Distance considerations and error performance . . . 177
9.2.4 Pragmatic TCM and two-stage decoding . . . 178

9.3 Multilevel coded modulation (MCM) 182
9.3.1 Constructions and multi-stage decoding 183
9.3.2 Unequal-error-protection with MCM 185
9.4 Bit-interleaved coded modulation (BICM) 191
9.4.1 Gray mapping 191
9.4.2 Metric generation: De-mapping 192
9.4.3 Interleaving 193
9.5 Turbo trellis-coded modulation (TTCM) 194
9.5.1 Pragmatic turbo TCM 194
9.5.2 Turbo TCM with symbol interleaving 194
9.5.3 Turbo TCM with bit interleaving 194

References 197

Appendix A Weight distributions of extended BCH codes 207
A.1 Length 8 207
A.2 Length 16 207
A.3 Length 32 208
A.4 Length 64 209
A.5 Length 128 211

Index 219

Preface

This book is the result of hundreds of emails from all over the world with questions on theory and applications of error correcting coding (ECC), from colleagues from both academia and industry. Most of the questions have been from engineers and computer scientists needing to select, implement or simulate a particular coding scheme. The questions were sparked by an ECC web site that was initially set up at Imai Laboratory at the Institute of Industrial Science, University of Tokyo, at the beginning of 1995. The reader will notice the absence of theorems and proofs in this text. The approach is to teach basic concepts by using simple examples. References to theoretical developments are made when needed. This book is intended to be a reference guide to error correcting coding techniques for graduate students and professionals interested in learning the basic techniques and applications of ECC. Computer programs that implement the basic encoding and decoding algorithms of practical coding schemes are available on a companion web site at:

`http://the-art-of-ecc.com`

This site is referred to as the "ECC web site" throughout the text. This book is unique in that it introduces the basic concepts of error correcting codes using simple illustrative examples. Computer programs, written in C language and available on the ECC web site, help to further illustrate the implementation of basic encoding and decoding algorithms of important coding schemes, such as convolutional codes, Hamming codes, BCH codes, Reed-Solomon codes and turbo codes, and their application in coded modulation systems. The material focuses on basic algorithms for analyzing and implementing ECC. There is a rich theory of ECC that will be touched upon, by referring to the appropriate material. There are many good books dealing with the theory of ECC, e.g., references [LC], [MS], [PW], [Blah], [Bos], [Wic], just to cite a few. Readers may wish to consult them before, during or after going through the material in this book. Each chapter describes, using simple and easy to follow numerical examples, the basic concepts of a particular coding or decoding scheme, rather than going into the detail of the theory behind it. Basic analysis tools are given throughout the book, to help in the assessment of the error performance of a particular ECC scheme, for some basic channel models. With the companion web site, this makes the book unique.

The book deals with the *art* of error correcting coding, in the sense that it addresses the need for selecting, implementing and simulating algorithms for encoding and decoding of codes for error correction and detection. The book is organized as follows. In the first chapter, the basic concepts of error correction, and coding and decoding techniques, are introduced. Chapter 2 deals with important and simple to understand families of codes, such as the Hamming, Golay and Reed-Muller codes. In Chapter 3, cyclic codes and the important family of BCH codes are described. Finite field arithmetic is introduced and basic decoding algorithms, such as

Berlekamp-Massey, Euclidean and PGZ, are described and easy to follow examples given to understand their operation. Chapter 4 deals with Reed-Solomon codes and errors-and-erasures decoding. A comprehensive treatment of the available algorithms is given, along with examples of their operation. In Chapter 5, binary convolutional codes are introduced. Focus in this chapter is on the understanding of the basic structure of these codes, along with a basic explanation of the Viterbi algorithm with Hamming metrics. Important implementation issues are discussed. In Chapter 6, several techniques for modifying a single code or combining several codes are given and illustrated by simple examples. Chapter 7 deals with soft-decision decoding algorithms, some of which have not yet received attention in the literature, such as a soft-output ordered statistics decoding algorithm. Moreover, Chapter 8 presents a unique treatment of turbo codes, both parallel concatenated and serial concatenated, and block product codes, from a coding theoretical perspective. In the same chapter, low-density parity check codes are examined. For all these classes of codes, basic decoding algorithms are described and simple examples are given. Finally, Chapter 9 deals with powerful techniques that combine error correcting coding with digital modulation, and several clever decoding techniques are described. A comprehensive bibliography is included, for readers who wish to learn more about the beautiful theory that makes it all work. It is my hope that this book will become a valuable and indispensable tool for both students and practitioners of this interesting, exciting and never-ending area of information theory.

I would like to thank the following persons for influencing this work. Professor Francisco Garcia Ugalde, Universidad Nacional Autónoma de México, for introducing me to the exciting world of error correcting codes. Parts of this book are based on my Bachelor's thesis under his direction. Professor Edward Bertram, University of Hawaii, for teaching me the basics of abstract algebra. Professor David Muñoz, Instituto Technológico y de Estudios Superiores de Monterrey, México, for his kindness and support. Professors Tadao Kasami, Hiroshima City University, Toru Fujiwara, University of Osaka, and Hideki Imai, University of Tokyo, for supporting my stays as a visiting academic researcher in Japan. Dan Luthi and Advait Mogre, LSI Logic Corporation, for many stimulating discussions and the opportunity to experience the process of putting ideas into silicon. Professor Marc P.C. Fossorier, University of Hawaii, for his help. My colleague Dr. Misa Mihaljević, Sony Computer Science Laboratories, for pointing out connections between decoding and cryptanalysis. I would also like to thank wholeheartedly Dr. Mario Tokoro, President of Sony Computer Science Laboratories, and Professor Ryuji Kohno, Yokohama National University, for making it possible for me to have a fine environment in which to write this book. In particular, I want to express my eternal gratitude to Professor Shu Lin, now at the University of California at Davis, who supported me when I was a graduate student in Hawaii, and encouraged me to continue my research in this fascinating topic. Last but not least, I want to thank the many students and colleagues who throughout the years listened to my lectures in Mexico, Japan and the U.S.A.

I dedicate this book to the memory of Richard W. Hamming, Claude Shannon and Gustave Solomon, three extraordinary gentlemen who greatly impacted the way people live and work today.

Robert H. Morelos-Zaragoza
Tokyo, Japan, April 2002.

Foreword

In modern digital communication and storage systems design, information theory is becoming increasingly important. The best example of this is the appearance and quick adoption of turbo and block product codes in many practical satellite and wireless communication systems. I am pleased to recommend this new book, authored by Dr. Robert Morelos-Zaragoza, to those who are interested in error correcting codes or have to apply them. The book introduces key concepts of error correcting coding (ECC) in a manner that is easy to understand. The material is logically well structured and presented using simple illustrative examples. This, together with the computer programs available on the web site, is a novel approach to teaching the basic techniques used in the design and application of error correcting codes.

One of the best features of the book is that it provides a natural introduction to the principles and decoding techniques of turbo codes, LDPC codes, and product codes, from an algebraic channel coding perspective. In this context, turbo codes are viewed as punctured product codes. With simple examples, the underlying ideas and structures used in the construction and iterative decoding of product codes are presented in an unparalleled manner. The detailed treatment of various algebraic decoding techniques for the correction of errors and erasures using Reed-Solomon codes is also worth a mention. On the applications of ECC in combined channel coding and digital modulation, or coded modulation, the author does a good job in introducing the basic principles that are used in the construction of several important classes of coded modulation systems.

I believe that practitioner engineers and computer scientists will find this book to be both a good learning tool and a valuable reference. The companion ECC web site is a unique feature that is not found anywhere else. Incidentally, this web site was born in my laboratory at the University of Tokyo in 1995, where Dr. Morelos-Zaragoza worked until June of 1997 and did a very good job as my associate researcher, writing many high-quality papers. Robert is polite, modest and hard-working, and is always friendly. In summary, I strongly recommend *The Art of Error Correcting Coding* as an excellent introductory and reference book on the principles and applications of error correcting codes.

Professor Hideki Imai
The University of Tokyo
Tokyo, Japan, April 2002

The ECC web site

The Art of Error Correcting Coding and its companion web site, the *ECC web site*, offer a new and unique approach to teaching the fundamental concepts of error correcting coding. The book explains in a clear and easy to understand manner, with simple illustrative examples, basic error correcting coding (ECC) techniques along with their decoding algorithms. Many practical ECC techniques are covered in the book, such as cyclic codes, BCH codes, RS codes, convolutional codes, turbo codes, product codes and low-density parity-check (LDPC) codes. In parallel with the tutorial treatment of the book, a companion web site provides readers with computer programs that implement decoding algorithms of the most important families of error correcting codes.

This is a novel *hands-on* method of teaching the *art* of ECC. Moreover, many of the computer programs on the web site can be used to simulate advanced techniques of error correcting coding, such as belief propagation (BP) decoding of LDPC codes and iterative decoding of product codes based on maximum-a-posteriori (MAP) decoding of the component codes. Also, programs are available on the ECC web site to simulate combinations of codes and digital modulation formats, and include trellis-coded modulation (TCM), multilevel coded modulation (MCM), bit-interleaved CM (BICM) and turbo TCM (T-TCM).

Highlights of *The Art of Error Correcting Coding* are the following:

⋆ **Comprehensive treatment of decoding procedures for BCH and RS codes**

⋆ **General decoders for RS codes**

Arbitrary shortening, arbitrary starting zero, errors-and-erasures decoding using the Berlekamp-Massey, Euclidean or Peterson-Gorenstein-Zierler (PGZ) algorithms.

⋆ **Techniques for modifying and combining linear codes**

Direct-sum, product, concatenation and generalized concatenation (GC).

⋆ **Reliability-based decoding of linear block codes**

Generalized minimum distance (GMD) decoding algorithm for RS codes, Chase algorithms and ordered-statistics decoding (OSD) algorithm for binary linear block codes. Viterbi decoding using a trellis.

⋆ **Soft-input soft-output (SISO) decoding of binary linear block codes**

This includes SO-Chase and SO-OSD algorithms, which have not received attention in other textbooks. Optimal MAP decoding and its approximations. Belief propagation decoding.

⋆ **Combined coding and modulation**

TCM, multilevel coding and unequal error protection (UEP), BICM and turbo TCM.

A companion web site for the book *The Art of Error Correcting Coding* has been set up and is located permanently at the following URL address:

`the-art-of-ecc.com`

The **ECC web site** contains computer programs written in C language to implement algorithms for encoding and decoding of important families of error correcting codes. The web site is maintained by the author, to ensure that the domain name remains unchanged. An important advantage of having a companion web site is that it allows the author to post update notes, new computer programs and simulation results relevant to the contents of the book. The computer programs in the ECC web site are organized in two ways: by topic and by function.

In the topical organization of the programs, the logical structure of the book is closely followed, going from simple syndrome-based decoding of linear block codes to more elaborate algebraic decoding over finite fields of BCH and Reed-Solomon codes, passing through Viterbi decoding of convolutional codes and decoding of combinations and constructions of codes, to iterative decoding of turbo and product codes, belief-propagation decoding of low-density parity-check codes and applications in coded modulation techniques. The index of programs by topic is summarized below.

1. **Linear block codes**
   - Computing the weight distribution
   - Bounds on error performance over AWGN, flat Rayleigh fading and BSC channels
2. **Hamming, Golay and Reed-Muller codes**
   - Hamming codes and syndrome-based decoding
   - Binary Golay (23,12,7) code and look-up table decoding
   - Extended Golay (24,12,8) code and syndrome-based decoding
3. **Binary cyclic codes and BCH codes**
   - Error-trapping decoding of binary cyclic codes
   - Berlekamp-Massey, Euclidean and PGZ algorithms
   - Chien search and error correction
   - Errors-and-erasures decoding
   - Error performance evaluation
4. **Reed-Solomon codes**
   - Berlekamp-Massey, Euclidean and PGZ algorithms

- Errors-and-erasures decoding
- Weight distributions and error performance evaluation

5. **Binary convolutional codes**
   - Weight enumeration sequences and error performance evaluation
   - Viterbi algorithm with Hamming metrics
   - Decoding of punctured convolutional codes
6. **Soft-decision decoding**
   - Viterbi algorithm with Euclidean metrics
   - Decoding binary linear block codes with a trellis
   - Chase type-II algorithm
   - Ordered statistics decoding (OSD)
   - GMD decoding
7. **Soft-input soft-output (SISO) algorithms**
   - MAP and log-MAP
   - Max-log-MAP and SOVA
   - SO-OSD and SO-Chase
8. **Iteratively decodable codes**
   - Parallel concatenation (turbo codes)
   - Serial concatenation (product codes)
   - Block product codes
   - LDPC codes: iterative decoding (bit-flip and belief-propagation)
9. **Coded modulation**
   - Trellis coded modulation (TCM) and pragmatic TCM
   - Multilevel coded modulation (MCM) and unequal-error-protection MCM
   - Bit-interleaved coded modulation (BICM)
   - Turbo TCM

The functional organization of the programs in the ECC web site is intended for readers who already know exactly what they are looking for. In particular, this classification of the programs is followed with respect to the decoding algorithms. The organization of the programs according to their functionality is summarized below.

1. **Basic performance analysis tools**
   - Bounds and approximations on the block and bit error rate
   - Data files with weight distributions of extended binary BCH codes
   - Program to compute the weight distribution of a code given its generator matrix
2. **Hard-decision decoding**
   - Syndrome decoding of short linear block codes

- Berlekamp-Massey algorithm-based decoding of RS and BCH codes
- Euclidean algorithm-based decoding of RS and BCH codes
- Peterson-Gorenstein-Zierler (PGZ) algorithm-based decoding of RS and BCH codes
- Viterbi algorithm

3. **Soft-decision decoding**
   - Viterbi algorithm for convolutional and linear block codes
   - Chase type-II algorithm
   - GMD algorithms
   - OSD algorithm
4. **SISO decoding**
   - Optimal MAP algorithm (also known as the BCJR[1] algorithm) and log-MAP algorithm
   - Max-log-MAP and soft-output Viterbi (SOVA) algorithms
   - SO Chase and SO-OSD algorithms for decoding linear block codes
5. **Iterative decoding**
   - Partial product ("parallel concatenation") codes
   - Full product ("serial concatenation") codes
   - Sum-product or BP algorithm for LDPC codes
6. **Combination of ECC and modulation**
   - MLD decoding of TCM and two-stage decoding of pragmatic TCM
   - Multistage decoding of multilevel codes
   - Applications of multilevel codes in unequal error protection
   - BICM
   - Turbo TCM

[1] BCJR stands for Bahl-Cocke-Jelinek-Raviv after the authors.

1

Introduction

The history of ECC started with the introduction of the Hamming codes [Ham], at about the same time as the seminal work of Shannon [Sha]. Shortly after, Golay codes were invented [Gol]. These first classes of codes are optimal, in a sense to be defined in a subsequent section.

Figure 1 shows the block diagram of a canonical digital communications/storage system. This is the famous *Figure 1* in most books on the theory of ECC. The information source and destination will include any source coding scheme matched to the nature of the information. The ECC encoder takes as input the information symbols from the source and adds redundant symbols to it, so that most of the errors — introduced in the process of modulating a signal, transmitting it over a noisy medium and demodulating it — can be corrected.

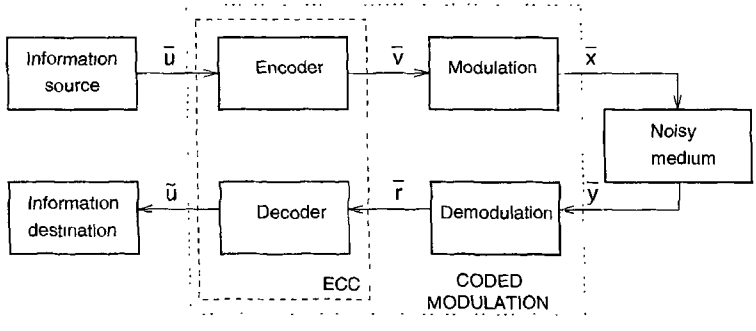


Figure 1 A canonical digital communications system.

Usually, the channel is assumed to be such that samples of an additive noise process are added to the modulated symbols (in their complex baseband representation). The noise samples are assumed to be independent from the source symbols. This model is relatively easy to track and includes additive white Gaussian noise (AWGN) channels, flat Rayleigh fading channels, and binary symmetric channels (BSC). At the receiver end, the ECC decoder utilizes the redundant symbols to correct channel errors. In the case of error detection, the ECC decoder can be thought of as a re-encoder of the received message and a check that the redundant symbols generated are the same as those received.

In classical ECC theory, the combination of modulation, noisy medium and demodulation was modeled as a *discrete memoryless channel* with input $\bar{v}$ and output $\bar{r}$. An example of

this is binary transmission over an AWGN channel, which is modeled as a *binary symmetric channel* (BSC) with a probability of channel error p — or transition probability — equal to the probability of a bit error for binary signalling over an AWGN,

$$p = Q\left(\sqrt{\frac{2E_b}{N_0}}\right), \tag{1.1}$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-z^2/2} dz, \qquad x \geq 0, \tag{1.2}$$

is the Gaussian Q-function[1], and E_b/N_0 is the signal-to-noise ratio (SNR) per bit. This case is treated later in this chapter.

In 1974, Massey [Mas3] suggested considering ECC and modulation as a single entity, known in modern literature as *coded modulation*. This approach provides a higher efficiency and coding gain[2] than the serial concatenation of ECC and modulation, by joint design of codes and signal constellations. Several methods of combining coding and modulation are covered in this book, including: Trellis-coded modulation (TCM) [Ung1] and multilevel coded modulation (MCM) [IH]. In a coded modulation system, the (soft-decision) channel outputs are processed directly by the decoder. In contrast, in a classical ECC system, the hard decision bits from the demodulator are fed to a binary decoder.

Codes can be combined in several ways. An example of *serial concatenation* (that is, concatenation in the classical sense) is the following. For years, the most popular concatenated ECC scheme has been the combination of an outer Reed-Solomon code, through intermediate interleaving, and an inner binary convolutional code. This scheme has been used in numerous applications, ranging from space communications to digital broadcasting of high definition television. The basic idea is that the soft-decision decoder of the convolutional code produces bursts of errors that can be broken into smaller pieces by the deinterleaving process and handled effectively by the Reed-Solomon decoder. Reed-Solomon codes are non-binary codes that work with symbols composed of several bits, and can deal with multiple bursts of errors. Serial concatenation has the advantage that it requires two separate decoders, one for the inner code and one for the outer code, instead of a single but very complex decoder for the overall code.

This book examines these types of ECC systems. First, basic code constructions and their decoding algorithms, in the Hamming space (that is, dealing with bits), are presented. Subsequently, the second part of the book introduces soft-decision decoding algorithms for binary transmission, that work over the Euclidean space and achieve a reduction in the required transmitted power per bit of at least 2 dB, compared with Hamming-space (hard-decision) decoders. Several kinds of soft-decision decoders are considered, with attention given to their algorithmic aspects (how they work) instead of their theoretical aspects (why they work). Finally, combinations of codes and interleaving for iterative decoding and combined coding and modulation are the topic of the last part of the book.

---

[1] Note that, in terms of the complementary error function, $Q(x) = \frac{1}{2}\text{erfc}(x/\sqrt{2})$.

[2] Coding gain is defined as the difference in signal-to-noise ratio between the coded system and an uncoded system with the same rate.

1.1 Error correcting coding: Basic concepts

All error correcting codes are based on the same basic principle: Redundancy is added to information in order to correct any errors that may occur in the process of storage or transmission. In a basic (and practical) form, redundant symbols are appended to information symbols to obtain a coded sequence or *codeword*. For the purpose of illustration, a codeword obtained by encoding with a *block code* is show in Figure 2. Such an encoding is said to be *systematic*. This means that the information symbols always appear in the first k positions of a codeword. The remaining $n - k$ symbols in a codeword are some function of the information symbols, and provide redundancy that can be used for error correction/detection purposes. The set of all code sequences is called an *error correcting code*, and will be denoted by C.

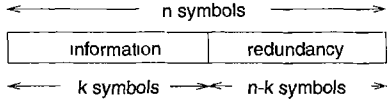


Figure 2 Systematic block encoding for error correction.

1.1.1 Block codes and convolutional codes

According to the manner in which redundancy is added to messages, ECC can be divided into two classes: block and convolutional. Both types of coding schemes have found practical applications. Historically, convolutional codes have been preferred, apparently because of the availability of the soft-decision Viterbi decoding algorithm and the belief for many years that block codes could not be efficiently decoded with soft-decisions. However, recent developments in the theory and design of soft-decision decoding algorithms for linear block codes have helped to dispel this belief. Moreover, the best ECC known to date (beginning of the twenty-first century) are block codes (irregular low-density parity-check codes).

Block codes process the information on a block-by-block basis, treating each block of information bits independently from others. In other words, block coding is a memoryless operation, in the sense that codewords are independent from each other. In contrast, the output of a convolutional encoder depends not only on the current input information, but also on previous inputs or outputs, either on a block-by-block or a bit-by-bit basis. For simplicity of exposition, we begin with a study of the structural properties of block codes. Many of these properties are common to both types of codes.

It should be noted that block codes have in fact memory, when encoding is thought of as a bit-by-bit process and within a codeword. Most recently the difference between block and convolutional codes has become less and less well defined, specially after recent advances in the understanding of the trellis structure of block codes and the tail-biting structure of some convolutional codes. Indeed, colleagues working on convolutional codes sometimes refer to block codes as "codes with time-varying trellis structure." Similarly, researchers working with block codes may consider convolutional codes as "codes with a regular trellis structure."

1.1.2 Hamming distance, Hamming spheres and error correcting capability

Consider an error correcting code C with binary elements. As mentioned above, block codes are considered for simplicity of exposition. In order to achieve error correcting capabilities, not all the 2^n possible binary vectors of length n are allowed to be transmitted. Instead, C is a subset of the n-dimensional binary vector space $V_2 = \{0,1\}^n$, such that its elements are as far apart as possible.

In the binary space V_2, *distance* is defined as the number of entries in which two vectors differ. Let $\bar{x}_1 = (x_{1,0}, x_{1,1}, \cdots, x_{1,n-1})$ and $\bar{x}_2 = (x_{2.0}, x_{2,1}, \cdots, x_{2,n-1})$ be two vectors in V_2. Then the *Hamming distance* between $\bar{x}_1$ and $\bar{x}_2$, denoted $d_H(\bar{x}_1, \bar{x}_2)$, is defined as

$$d_H(\bar{x}_1, \bar{x}_2) = \left| \{i : x_{1,i} \neq x_{2,i}, \quad 0 \leq i < n\} \right|, \tag{1.3}$$

where $|A|$ denotes the number of elements in (or the cardinality of) a set A.

Given a code C, its *minimum Hamming distance*, d_{min}, is defined as the minimum Hamming distance among all possible distinct pairs of codewords in C,

$$d_{min} = \min_{\bar{v}_1, \bar{v}_2 \in C} \{d_H(\bar{v}_1, \bar{v}_2) | \bar{v}_1 \neq \bar{v}_2\}. \tag{1.4}$$

Throughout the book, (n, k, d_{min}) is used to denote the parameters of a block code of length n, that encodes messages of length k bits and has a minimum Hamming distance d_{min}. The assumption is made that the size of the code is $|C| = 2^k$.

Example 1 The simplest error correcting code is a binary repetition code of length 3. It repeats each bit three times, so that a '0' is encoded onto the vector (000) and a '1' onto the vector (111). Since the two codewords differ in all three positions, the Hamming distance between them is equal to three. Figure 3 is a pictorial representation of this code. The 3-dimensional binary space corresponds to the set of $2^3 = 8$ vertices of the three-dimensional unit-volume cube. The Hamming distance between codewords (000) and (111) equals the number of edges in a path between them. This is equivalent to the number of coordinates that one needs to change to convert (000) into (111), or vice versa. Thus $d_H((000), (111)) = 3$. Since there are only two codewords in this case, $d_{min} = 3$.

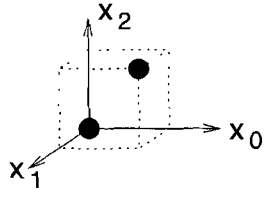


Figure 3 A (3,1,3) repetition code in a 3-dimensional binary vector space.

The binary vector space V_2 is also known as a *Hamming space*. Let $\bar{v}$ denote a codeword of an error correcting code C. A *Hamming sphere* $S_t(\bar{v})$, of *radius* t and centered around $\bar{v}$, is the set of vectors in V_2 at a distance less than or equal to t from the center $\bar{v}$,

$$S_t(\bar{v}) = \{\bar{x} \in V_2 | d_H(\bar{x}, \bar{v}) \leq t\}. \tag{1.5}$$

Note that the size of (or the number of codewords in) $S_t(\bar{v})$ is given by the following expression

$$|S_t(\bar{v})| = \sum_{i=0}^{t} \binom{n}{i}. \tag{1.6}$$

Example 2 Figure 4 shows the Hamming spheres of radius $t = 1$ around the codewords of the $(3, 1, 3)$ binary repetition code.

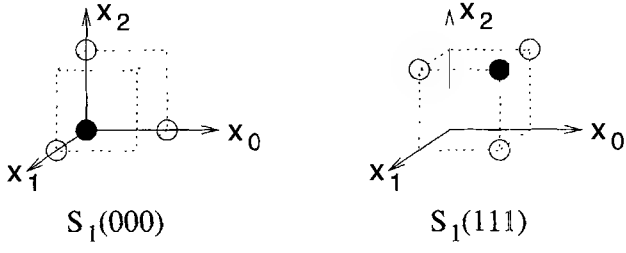


Figure 4 Hamming spheres of radius $t = 1$ around the codewords of the (3,1,3) binary repetition code.

Note that the Hamming spheres for this code are disjoint, that is, there is no vector in V_2 (or vertex in the unit-volume three-dimensional cube) that belongs to both $S_1(000)$ and $S_1(111)$. As a result, if there is a change in any one position of a codeword $\bar{v}$, then the resulting vector will still lie inside a Hamming sphere centered at $\bar{v}$. This concept is the basis of understanding and defining the error correcting capability of a code C.

The *error correcting capability*, t, of a code C is the largest radius of Hamming spheres $S_t(\bar{v})$ around all the codewords $\bar{v} \in C$, such that for all different pairs $\bar{v}_i, \bar{v}_j \in C$, the corresponding Hamming spheres are disjoint, i.e.,

$$t = \max_{\bar{v}_i, \bar{v}_j \in C} \{\ell | S_\ell(\bar{v}_i) \cap S_\ell(\bar{v}_j) = \emptyset, \quad \bar{v}_i \neq \bar{v}_j\}. \tag{1.7}$$

In terms of the minimum distance of C, d_{min}, an equivalent and more common definition is

$$t = \lfloor (d_{min} - 1)/2 \rfloor, \tag{1.8}$$

where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x.

Note that in order to compute the minimum distance d_{min} of a block code C, in accordance with Equation (1.4), a total of (at most) $2^k(2^k - 1)$ distances between distinct pairs of codewords are needed. This is practically impossible even for codes of relatively modest size, say, $k = 50$. One of the advantages of *linear* block codes is that the computation of d_{min} requires one only to know the *Hamming weight* of all $2^k - 1$ nonzero codewords.

1.2 Linear block codes

As mentioned above, finding a good code means finding a subset of V_2 with elements as far apart as possible. This is very difficult. In addition, even when such a set is found, there is still the problem of *how to assign codewords to information messages.*

Linear codes are *vector subspaces* of V_2. This means that encoding can be accomplished by matrix multiplications. In terms of digital circuitry, simple encoders can be built using exclusive OR's, AND gates and D flip-flops. In this chapter, the binary vector space operations of sum and multiplication are meant to be the output of exclusive-OR (modulo 2 addition) and AND gates, respectively. The tables of addition and multiplication for binary elements are:

a	b	$a+b$	$a \cdot b$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

which correspond, as mentioned previously, to the outputs of a binary "X-OR" logic gate and a binary "AND" logic gate, respectively.

1.2.1 Generator and parity-check matrices

Let C denote a binary linear (n, k, d_{min}) code. Since C is a k-dimensional vector subspace, it has a *basis*, say $\{\bar{v}_0, \bar{v}_1, \cdots, \bar{v}_{k-1}\}$, such that any codeword $\bar{v} \in C$ can be represented as a linear combination of the elements in the basis:

$$\bar{v} = u_0\bar{v}_0 + u_1\bar{v}_1 + \cdots + u_{k-1}\bar{v}_{k-1}, \tag{1.9}$$

where $u_i \in \{0, 1\}$, $1 \leq i < k$. Equation (1.9) can be written in terms of a *generator matrix* G and a *message vector*, $\bar{u} = (u_0, u_1, \cdots, u_{k-1})$, as follows:

$$\bar{v} = \bar{u}G, \tag{1.10}$$

where

$$G = \begin{pmatrix} \bar{v}_0 \\ \bar{v}_1 \\ \vdots \\ \bar{v}_{k-1} \end{pmatrix} = \begin{pmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,n-1} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k-1,0} & v_{k-1,1} & \cdots & v_{k-1,n-1} \end{pmatrix}. \tag{1.11}$$

Since C is a k-dimensional vector space in V_2, there is an $(n-k)$-dimensional *dual space* $C^\top$, generated by the rows of a matrix H, called the *parity-check matrix*, such that $GH^\top = 0$, where $H^\top$ denotes the transpose of H. In particular, note that for any codeword $\bar{v} \in C$,

$$\bar{v}H^\top = \bar{0}. \tag{1.12}$$

Equation (1.12) is of fundamental importance in *decoding of linear codes*, as will be shown in section 1.3.2.

A linear code $C^\perp$ that is generated by H is a binary linear $(n, n-k, d^\perp_{min})$ code, called the *dual code* of C.

1.2.2 The weight is the distance

As mentioned in section 1.1.2, a nice feature of linear codes is that computing the minimum distance of the code amounts to computing the minimum Hamming weight of its nonzero

codewords. In this section, this fact is shown. First, define the *Hamming weight*, $\mathrm{wt}_H(\bar{x})$, of a vector $\bar{x} \in V_2$ as the number of nonzero elements in $\bar{x}$. From the definition of the Hamming distance, it is easy to see that $\mathrm{wt}_H(x) = d_H(\bar{x}, \bar{0})$. For a binary linear code C, note that the distance

$$d_H(\bar{v}_1, \bar{v}_2) = d_H(\bar{v}_1 + \bar{v}_2, \bar{0}) = \mathrm{wt}_H(\bar{v}_1 + \bar{v}_2). \tag{1.13}$$

Finally, by linearity, $\bar{v}_1 + \bar{v}_2 \in C$. As a consequence, the minimum distance of C can be computed by finding the minimum Hamming weight among the $2^k - 1$ nonzero codewords. This is simpler than the brute force search among all pairs of codewords, although still a considerable task even for codes of modest size (or dimension k).

1.3 Encoding and decoding of linear block codes

1.3.1 Encoding with G and H

Equation (1.10) gives an encoding rule for linear block codes that can be implemented in a straightforward way. If encoding is to be systematic, then the generator matrix G of a linear block (n, k, d_{min}) code C can be brought to a *systematic form*, G_{sys}, by elementary row operations and/or column permutations. G_{sys} is composed of two sub-matrices: The k-by-k identity matrix, denoted I_k, and a k-by-$(n-k)$ *parity sub-matrix* P, such that

$$G_{sys} = (I_k | P), \tag{1.14}$$

where

$$P = \begin{pmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-k\ 1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n\ -k-1} \end{pmatrix} \tag{1.15}$$

Since $GH^\top = 0$, it follows that the systematic form, H_{sys}, of the parity-check matrix is

$$H_{sys} = (P^\top | I_{n-k}). \tag{1.16}$$

Example 3 Consider a binary linear $(4, 2, 2)$ code with generator matrix

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

To bring G into systematic form, permute (exchange) the second and fourth columns and obtain

$$G_{sys} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Thus the parity-check sub-matrix is given by

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

It is interesting to note that in this case, the relation $P = P^\top$ holds[3]. From (1.16) it follows

[3] In this case, the code in question is referred to as a *self-dual code*. See also Section 2.2.3.

that the systematic form of the parity-check matrix is

$$H_{sys} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

In the following let $\bar{u} = (u_0, u_1, \cdots, u_{k-1})$ denote an information message to be encoded and $\bar{v} = (v_0, v_1, \cdots, v_{n-1})$ the corresponding codeword in C.

If the parameters of C are such that $k < (n-k)$, or equivalently the *code rate* $k/n < 1/2$, then encoding with the generator matrix is the most economical. The cost considered here is in terms of binary operations. In such case

$$\bar{v} = \bar{u}G_{sys} = (\bar{u}, \bar{v}_p), \tag{1.17}$$

where $\bar{v}_p = \bar{u}P = (v_k, v_{k+1}, \cdots, v_{n-1})$ represents the parity-check part of the codeword.

However, if $k > (n-k)$, or $k/n > 1/2$, then alternative encoding with the parity-check matrix H requires less number of computations. In this case, we have encoding based on Equation (1.12), $(\bar{u}, \bar{v}_p)H^\top = 0$, such that the $(n-k)$ parity-check positions $v_k, v_{k+1}, \cdots, v_{n-1}$ are obtained as follows:

$$v_j = u_0 p_{0,j} + u_1 p_{1,j} + \cdots + u_{k-1} p_{k-1,j}, \qquad k \leq j < n. \tag{1.18}$$

Stated in other terms, the systematic form of a parity-check matrix of a linear code has as entries of its rows the coefficients of the parity-check equations, from which the values of the redundant positions are obtained. This fact will be used when low-density parity-check codes are presented, in Section 8.3.

Example 4 Consider the binary linear $(4, 2, 2)$ code from Example 3. Let messages and codewords be denoted by $\bar{u} = (u_0, u_1)$ and $\bar{v} = (v_0, v_1, v_2, v_3)$, respectively. From Equation (1.18), we have that

$$\begin{aligned} v_2 &= u_0 + u_1 \\ v_3 &= u_0 \end{aligned}$$

The correspondence between the $2^2 = 4$ two-bit messages and codewords is as follows:

$$\begin{aligned} (00) &\mapsto (0000) \\ (01) &\mapsto (0110) \\ (10) &\mapsto (1011) \\ (11) &\mapsto (1101) \end{aligned} \tag{1.19}$$

1.3.2 Standard array decoding

In this section, a decoding procedure is presented that finds the closest codeword $\bar{v}$ to a received noisy word $\bar{r} = \bar{v} + \bar{e}$. The *error vector* $\bar{e} \in \{0,1\}^n$ is produced by a BSC, as depicted in Figure 5. It is assumed that the crossover probability (or BSC parameter) p is such that $p < 1/2$.

A *standard array* [Sle] for a binary linear (n, k, d_{min}) code C is a table of all possible received vectors $\bar{r}$ arranged in such a way that the closest codeword $\bar{v}$ to $\bar{r}$ can be read out.

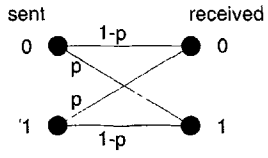


Figure 5 A binary symmetric channel model.

Table 1 The standard array of a binary linear block code.

$\bar{s}$	$\bar{u}_0 = \bar{0}$	$\bar{u}_2$	$\cdots$	$\bar{u}_{k-1}$
$\bar{0}$	$\bar{v}_0 = \bar{0}$	$\bar{v}_1$	$\cdots$	$\bar{v}_{2^k-1}$
$\bar{s}_1$	$\bar{e}_1$	$\bar{e}_1 + \bar{v}_1$	$\cdots$	$\bar{e}_1 + \bar{v}_{2^k-1}$
$\bar{s}_2$	$\bar{e}_2$	$\bar{e}_2 + \bar{v}_1$	$\cdots$	$\bar{e}_2 + \bar{v}_{2^k-1}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\bar{s}_{2^{n-k}-1}$	$\bar{e}_{2^{n-k}-1}$	$\bar{e}_{2^{n-k}-1} + \bar{v}_1$	$\cdots$	$\bar{e}_{2^{n-k}-1} + \bar{v}_{2^k-1}$

The standard array contains 2^{n-k} rows and $2^k + 1$ columns. The entries of the rightmost 2^k columns of the array contain all the vectors in $V_2 = \{0, 1\}^n$.

In order to describe the decoding procedure, the concept of *syndrome* is needed. The syndrome of a word in V_2 is defined from Equation (1.12) as

$$\bar{s} = \bar{r}H^\top, \tag{1.20}$$

where H is the parity-check matrix of C. That $\bar{s}$ is indeed a set of symptoms that indicate errors is shown as follows. Suppose that a codeword $\bar{v} \in C$ is transmitted over a BSC and received as $\bar{r} = \bar{v} + \bar{e}$. The syndrome of $\bar{r}$ is

$$\bar{s} = \bar{r}H^\top = (\bar{v} + \bar{e})H^\top = \bar{e}H^\top, \tag{1.21}$$

where to obtain the last equality Equation (1.12) has been used. Therefore, the computation of the syndrome can be thought of as a *linear transformation* of an error vector.

Standard array construction procedure

1. As the first row, in the positions corresponding to the 2^k rightmost columns, enter all the codewords of C, beginning with the all-zero codeword in the leftmost position. In the position corresponding to the first column, enter the all-zero syndrome. Let $j = 0$.
2. Let $j = j + 1$. Find the smallest Hamming weight word $\bar{e}_j$ in V_2, not in C, and not included in previous rows. The corresponding syndrome $\bar{s}_j = \bar{e}_j H^\top$ is the first (rightmost) entry of the row. The 2^k remaining entries in that row are obtained by adding $\bar{e}_j$ to all the entries in the first row (the codewords of C).
3. Repeat the previous step until all vectors in V_2 are included in the array. Equivalently, let $j = j + 1$. If $j < 2^{n-k}$, then repeat previous step, otherwise stop.

Example 5 The standard array of the binary linear $(4, 2, 2)$ code is the following:

$\bar{s}$	00	01	10	11
00	0000	0110	1011	1101
11	1000	1110	0011	0101
10	0100	0010	1111	1001
01	0001	0111	1010	1100

Decoding with the standard array proceeds as follows. Let $\bar{r} = \bar{v} + \bar{e}$ be the received word. Find the word in the array and output as decoded message $\bar{u}$ the header of the column in which $\bar{r}$ lies. Conceptually, this process requires storing the entire array and matching the received word to an entry in the array.

However, a simplified decoding procedure can be found by noticing that every row in the array has the same syndrome. Each row of the array, denoted Row_i, for $0 \leq i < 2^{n-k}$, is a *coset* of C, such that $\mathrm{Row}_i = \left\{\bar{e}_i + \bar{v} \middle| \bar{v} \in C\right\}$. The vector $\bar{e}_i$ is known as the *coset leader*.

The syndrome of the elements in the i-th row is given by

$$\bar{s}_i = (\bar{e}_i + \bar{v})H^\top = \bar{e}_i H^\top, \tag{1.22}$$

which is *independent* of the particular choice of $\bar{v} \in C$. The simplified decoding procedure is: Compute the syndrome of the received word $\bar{r} = \bar{e}_{i'} + \bar{v}$,

$$\bar{s}_{i'} = (\bar{e}_{i'} + \bar{v})\, H^\top = \bar{e}_{i'} H^\top,$$

and find $\bar{s}_{i'}$ in the leftmost column of the standard array. Then read out the value of $\bar{e}_{i'}$, from the second column, an add it to the received word to obtain the closest codeword $\bar{v}' \in C$ to $\bar{r}$. Therefore instead of $n \times 2^n$ bits, standard array decoding can be implemented with an array of $n \times 2^{n-k}$ bits.

Example 6 Consider again the binary linear $(4, 2, 2)$ code from Example 3. Suppose that the codeword $\bar{v} = (0110)$ is transmitted and that $\bar{r} = (0010)$ is received. Then the syndrome is

$$\bar{s} = \bar{r}H^\top = (0010)\begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = (1 \quad 0)\,.$$

From the standard array of the code, the corresponding coset leader $\bar{e}' = (0100)$ is found, and therefore the estimated codeword is $\bar{v}' = \bar{r} + e' = (0010) + (0100) = (0110)$. One error has been corrected! This may sound strange, since the minimum distance of the code is only two and thus according to (1.8) single error correction is impossible. However, this can be explained by looking again at the standard array of this code (Example 5 above). Note that the third row of the array contains two distinct binary vectors of weight one. This means that only three out of a total of four single-error patterns can be corrected. The error above is one of those correctable single-error patterns.

It turns out that this $(4, 2, 2)$ code is the simplest instance of a linear *unequal error protection* (LUEP) code [WV, Van]. This LUEP code has a *separation vector* $\bar{s} = (3, 2)$, which means that the minimum distance between any two codewords for which the *first* message bit differs is at least three and that for the *second* message bit is at least two.

If encoding is *systematic*, then the above procedure gives the estimated message $\bar{u}'$ in the first k positions of $\bar{v}'$. This is a plausible reason for having a systematic encoding.

1.3.3 Hamming spheres, decoding regions and the standard array

The standard array is also a convenient way of understanding the concept of Hamming sphere and error correcting capability of a linear code C, introduced in Section 1.1.2.

By construction, note that the 2^k rightmost columns of the standard array, denoted Col_j, for $1 \leq j \leq 2^k$, contain a codeword $\bar{v}_j \in C$ and a set of $2^{n-k}-1$ words at the smallest Hamming distance from $\bar{v}_j$, that is,

$$\mathrm{Col}_j = \left\{ \bar{v}_j + \bar{e}_i \middle| \bar{e}_i \in \mathrm{Row}_i, \quad 0 \leq i < 2^{n-k} \right\}. \tag{1.23}$$

The sets Col_j are the *decoding regions*, in the Hamming space, around each codeword $\bar{v}_j \in C$, for $0 \leq j \leq 2^k - 1$. This is to say that if codeword $\bar{v}_j \in C$ is transmitted over a BSC and the received word $\bar{r}$ lies in the set Col_j, then it will be successfully decoded into $\bar{v}_j$.

Hamming bound

The set Col_j and the error correcting capability t of code C are related by the Hamming sphere $S_t(\bar{v}_j)$: A binary linear (n, k, d_{min}) code C has decoding regions Col_j that properly contain Hamming spheres $S_t(\bar{v}_j)$, i.e., $S_t(\bar{v}_j) \subseteq \mathrm{Col}_j$.

By noticing that the size of Col_j is 2^{n-k}, and using Equation (1.6), we obtain the celebrated *Hamming bound*

$$\sum_{i=0}^{t} \binom{n}{i} \leq 2^{n-k}. \tag{1.24}$$

The Hamming bound has several combinatorial interpretations. One of them is:

> *The number of syndromes, 2^{n-k}, must be greater than or equal to the number of correctable error patterns, $\sum_{i=0}^{t} \binom{n}{i}$.*

Example 7 The binary repetition $(3, 1, 3)$ code has generator matrix $G = (1 \quad 1 \quad 1)$ and parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

Accordingly, its standard array is the following:

$\bar{s}$	0	1
00	000	111
11	100	011
10	010	101
01	001	110

The four vectors in the second column of the array (i.e., the coset leaders) are the elements of the Hamming sphere $S_1(000)$ in Figure 4, which consists of all binary vectors of length three with Hamming weight less that or equal to one. Similarly, the entries of the third (rightmost) column of the array are the elements of $S_1(111)$. For this code, the Hamming bound (1.24) holds with equality.

Block codes satisfying the bound (1.24) are said to be *perfect codes*. The only perfect nontrivial codes are the binary Hamming $(2^m - 1, 2^m - m - 1, 3)$ codes, the nonbinary

Hamming $(\frac{q^m-1}{(q-1)}, \frac{q^m-1}{(q-1)} - m - 1, 3)$ codes, $q > 2$, the repetition $(n, 1, n)$ codes, the parity-check $(n, n-1, 2)$ codes, the binary Golay $(23, 12, 7)$ code and the ternary Golay $(11, 6, 5)$ code. The extended codes (obtained by appending an overall parity-check bit) of the Hamming and Golay codes are also perfect.

For nonbinary linear codes, defined over a field of q elements, with $q = p^m$ and $p > 2$ a prime number, the Hamming bound becomes

$$\sum_{i=0}^{n} \binom{n}{i} (q-1)^i \leq q^{n-k}. \tag{1.25}$$

1.4 Weight distribution and error performance

When selecting a particular coding scheme, it is important to assess its error performance. There are several measures of the performance of an ECC scheme. In this section, expressions for linear codes are introduced, for three basic channel models: The BSC model, the additive white Gaussian noise (AWGN) channel model and the flat Rayleigh fading channel model.

1.4.1 Weight distribution and undetected error probability over a BSC

The *weight distribution* $\mathcal{W}(C)$ of an error correcting code C, is defined as the set of $n+1$ integers $\mathcal{W}(C) = \{A_i,\ 0 \leq i \leq n\}$, such that there are A_i codewords of Hamming weight i in C, for $i = 0, 1, \cdots, n$.

An expression for the probability of undetected error of a linear code over a BSC is derived next. First, note that the Hamming weight of a word $\bar{v}$, $\mathrm{wt}_H(\bar{v})$ equals the Hamming distance to the all-zero word, $\mathrm{wt}_H(\bar{v}) = d_H(\bar{v}, \bar{0})$. Also, as noted before, the Hamming distance between any two codewords $\bar{v}_1, \bar{v}_2$ in a linear code C equals the Hamming weight of their difference,

$$d_H(\bar{v}_1, \bar{v}_2) = d_H(\bar{v}_1 + \bar{v}_2, \bar{0}) = \mathrm{wt}_H(\bar{v}_1 + \bar{v}_2) = \mathrm{wt}_H(\bar{v}_3).$$

where, by linearity of C, $\bar{v}_3 \in C$.

The *probability of an undetected error*, denoted $P_u(C)$, is the probability that the received word differs from the transmitted codeword but the syndrome equals zero. That is,

$$\bar{s} = (\bar{v} + \bar{e})H^\top = \bar{e}H^\top = 0 \quad \Longleftrightarrow \quad \bar{e} \in C.$$

Therefore, the probability that the syndrome of the received word is zero equals the probability that an error vector is a nonzero codeword in C.

With transmission over a BSC, the probability that the error vector $\bar{e}$ has weight i equals the probability that i bits are in error and that the remaining $n-i$ bits are correct. Let $P(e, i)$ denote this probability. Then

$$P(\bar{e}, i) = p^i (1-p)^{n-i}.$$

For an undetected error to occur, the error vector $\bar{e}$ must be a nonzero codeword. There are A_i vectors of weight i in C. It follows that

$$P_u(C) = \sum_{i=d_{min}}^{n} A_i\, P(\bar{e}, i) = \sum_{i=d_{min}}^{n} A_i\, p^i (1-p)^{n-i}. \tag{1.26}$$

Equation (1.26) gives the exact value of $P_u(C)$. Unfortunately, for most codes of practical interest the weight distribution $\mathcal{W}(C)$ is unknown. In these cases, using the fact that the number of codewords of weight i is less than or equal to the total number of words of weight i in the binary space V_2, the following upper bound is obtained:

$$P_u(C) \leq \sum_{i=d_{min}}^{n} \binom{n}{i} p^i (1-p)^{n-i}. \tag{1.27}$$

Expressions (1.26) and (1.27) are useful when an ECC scheme is applied for error detection only, such as in communication systems with feedback and ARQ. When a code is employed for error correction purposes, the expressions derived in the next sections are useful.

Example 8 For the binary linear $(4, 2, 2)$ code of Example 4, $\mathcal{W}(C) = (1, 0, 1, 2, 0)$. Therefore, Equation (1.26) gives

$$P_u(C) = p^2(1-p)^2 + 2\,p^3(1-p).$$

Figure 6 shows a plot of $P_u(C)$ compared with the upper bound in the right-hand side (RHS) of (1.27).

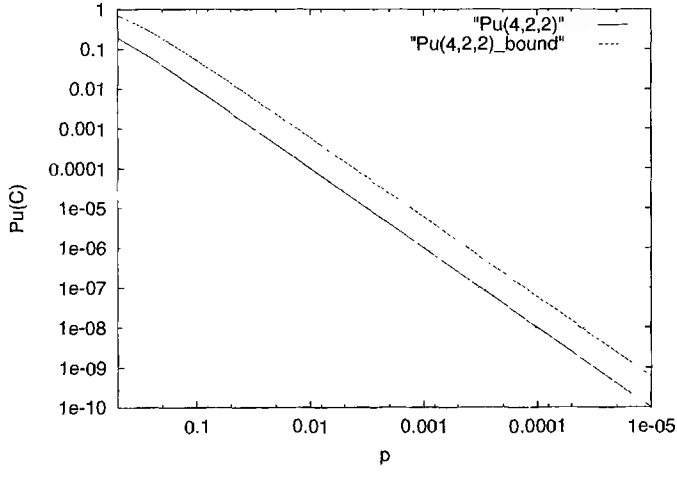


Figure 6 Exact value and upper bound on the probability of undetected error for a binary linear (4,2,2) code over an BSC.

1.4.2 Performance bounds over BSC, AWGN and fading channels

The purpose of this section is to introduce basic channel models that will be considered in the book, as well as the corresponding expressions on the error correction performance of linear codes. Error correction for the BSC is considered first.

The BSC model

For a binary linear code C, as explained in the previous section, a decoding procedure using the standard array will decode a received vector into the closest codeword. A decoding error will be made whenever the received vectors falls outside the correct decoding region.

Let L_i denote the number of coset leaders of weight i in the standard array of a linear code C. The *probability of a correct decoding* equals the probability that an error vector is a coset leader and given by

$$P_c(C) = \sum_{i=0}^{\ell} L_i \, p^i (1-p)^{n-i}, \tag{1.28}$$

where ℓ is the largest Hamming weight of a coset leader $\bar{e}$ in the standard array. For perfect codes, $\ell = t$, and

$$L_i = \binom{n}{i}, \quad 0 \leq i \leq t,$$

such that, from the Hamming bound (1.24),

$$\sum_{i=0}^{\ell} L_i = \sum_{i=0}^{t} \binom{n}{i} = 2^{n-k}.$$

For binary codes in general, Equation (1.28) becomes a lower bound on $P_c(C)$, since there exist coset leaders of weight greater than t.

The *probability of incorrect decoding*, denoted by $P_e(C)$, also known as the *probability of a decoding error*, is equal to the probability of the complement set of the event of correct decoding, i.e., $P_e(C) = 1 - P_c(C)$. From Equation (1.28), it follows that

$$P_e(C) = 1 - \sum_{i=0}^{\ell} L_i \, p^i (1-p)^{n-i}. \tag{1.29}$$

Finally, based on the above discussion for $P_c(C)$, the following upper bound is obtained,

$$P_e(C) \leq 1 - \sum_{i=0}^{t} \binom{n}{i} p^i (1-p)^{n-i}, \tag{1.30}$$

which can also be expressed as

$$P_e(C) \leq \sum_{i=t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}, \tag{1.31}$$

with equality if, and only if, code C is perfect (satisfies the Hamming bound with equality).

Example 9 Figure 7 shows the values of $P_e(C)$ from (1.31), as a function of the crossover probability p of a BSC, for the binary repetition (3,1,3) code.

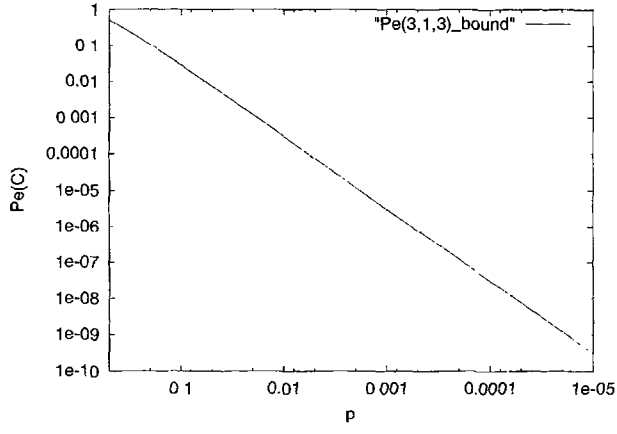


Figure 7 Probability of a decoding error for the binary repetition (3,1,3) code.

The AWGN channel model

Perhaps the most important channel model in digital communications is the additive white Gaussian noise (AWGN) channel. In this section, expressions are given for the probability of a decoding error and of a bit error for linear block codes over AWGN channels. Although similar expressions hold for convolutional codes, for clarity of exposition, they are introduced together with the discussion on soft-decision decoding with the Viterbi algorithm and coded modulation in subsequent chapters. The following results constitute valuable analysis tools for the error performance evaluation of binary coded schemes over AWGN channels.

Consider a binary transmission system, with coded bits in the set $\{0, 1\}$ mapped onto real values $\{+1, -1\}$, respectively, as illustrated in Figure 8. In the following, vectors are n-dimensional and the following notation is used to denote a vector: $\bar{x} = (x_0, x_1, \cdots, x_{n-1})$. The conditional probability density function (pdf) of the channel output sequence $\bar{y}$, given the input sequence $\bar{x}$ is given by

$$p(\bar{y}|\bar{x}) = p_{\bar{n}}(\bar{y} - \bar{x}) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(y_i - x_i)^2}{N_0}}, \tag{1.32}$$

where $p_{\bar{n}}(\bar{n})$ is the pdf of n statistically independent and identically distributed (i.i.d.) noise samples, each of which is Gaussian distributed with mean $\mu_n = 0$ and variance $\sigma_n = N_0/2$, and N_0 is the one-sided power spectral density of the noise. It is easy to show that *maximum-likelihood decoding* (MLD) of a linear code C over this channel selects a sequence $\hat{x}$ that minimizes the *squared Euclidean distance* between the received sequence $\bar{y}$ and $\hat{x}$,

$$D^2(\hat{x}, \bar{y}) \triangleq \sum_{i=0}^{n-1} (x_i - y_i)^2. \tag{1.33}$$

See, e.g., [WJ], [Wil] and [BM]. It should be noted that a decoder using Equation (1.33) as a *metric* is referred to as a *soft-decision decoder*, independently of whether or not MLD is performed. In Chapter 7, soft-decision decoding methods are considered.

The probability of a decoding error with MLD, denoted $P_e(C)$, is equal to the probability that a coded sequence $\bar{x}$ is transmitted and the noise vector $\bar{n}$ is such that the received sequence

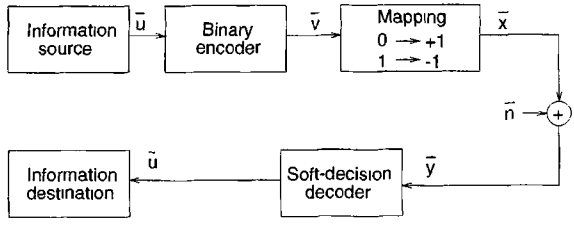


Figure 8 Binary coded transmission system over an AWGN channel.

$\bar{y} = \bar{x} + \bar{n}$ is closer to a different coded sequence $\hat{x} \in C$, $\hat{x} \neq \bar{x}$. For a linear code C, it can be assumed that the all-zero codeword is transmitted. Then $P_e(C)$ can be upper bounded, based on the *union bound* [Cla] and the *weight distribution* $W(C)$, as follows:

$$P_e(C) \leq \sum_{w=d_{min}}^{n} A_w \, Q\left(\sqrt{2wR\frac{E_b}{N_0}}\right), \tag{1.34}$$

where $R = k/n$ is the code rate, E_b/N_0 is the *energy per bit-to-noise ratio* (or SNR per bit) and $Q(x)$ is given by (1.2).

Figure 9 shows the evaluation of expressions for hard-decision decoding (1.30) and soft-decision decoding (1.34) for the binary (3,1,3) code. *Hard-decision decoding* means using a decoder for the BSC which is fed by the outputs from a binary demodulator. The equivalent BSC has a crossover probability equal to [Pro, WJ]

$$p = Q\left(\sqrt{2R\frac{E_b}{N_0}}\right).$$

Note that in this particular case, since the code is perfect and contains only two codewords, both expressions are exact, not upper bounds. Figure 9 also serves to illustrate the fact that soft-decision decoding performs better than hard-decision decoding, in the sense of requiring less transmitted power to achieve the same $P_e(C)$. The difference (in dB) between the corresponding SNR per bit is commonly referred to as *coding gain*.

In [FLR], the authors show that for *systematic* binary linear codes with binary transmission over an AWGN channel, the *probability of a bit error*, denoted $P_b(C)$, has the following upper bound:

$$P_b(C) \leq \sum_{w=d_{min}}^{n} \frac{wA_w}{n} Q\left(\sqrt{2wR\frac{E_b}{N_0}}\right). \tag{1.35}$$

Interestingly, besides the fact that the above bound holds only for systematic encoding, the results in [FLR] show that *systematic encoding minimizes the probability of a bit error*. This means that systematic encoding is not only desirable, but actually optimal in the above sense.

Example 10 Consider a linear binary (6,3,3) code with generator and parity-check matrices

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}, \qquad H = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix},$$

respectively. The weight distribution of this code is $W(C) = \{1, 0, 0, 4, 3, 0, 0\}$, which can be verified by direct computation of all the codewords $\bar{v} = (\bar{u}, \bar{v}_p)$:

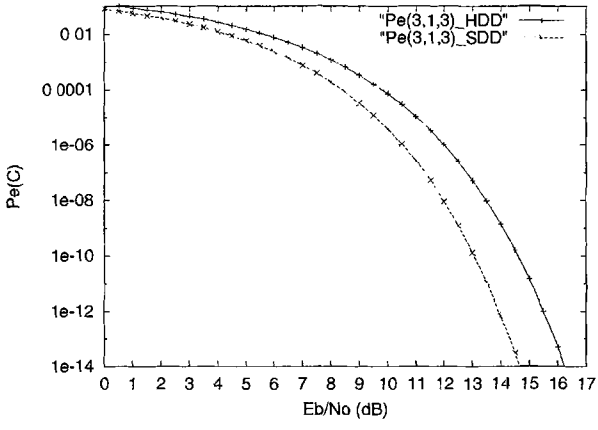


Figure 9 Probability of a decoding error for hard-decision decoding (Pe(3,1,3)_HDD) and soft-decision (Pe(3,1,3)_SDD) decoding of a binary (3,1,3) code. Binary transmission over an AWGN channel.

$\bar{u}$	$\bar{v}_p$
000	000
001	101
010	011
011	110
100	110
101	011
110	101
111	000

In this particular case, MLD can be performed by simply computing the squared Euclidean distance, Equation (1.33), between the received sequence and all of the eight candidate codewords. The decoded codeword is selected as that with the smallest distance. In Chapter 7, efficient methods for MLD and near-MLD of linear block codes are presented. Figure 10 shows simulations and union bounds with hard-decision and soft-decision MLD decoding with binary transmission over an AWGN channel.

The flat Rayleigh fading channel model

Another important channel model is that of flat Rayleigh fading. Fading occurs in wireless communication systems in the form of a time-varying distortion of the transmitted signal. In this book, we consider the case of flat Rayleigh fading. The term "flat" refers to the fact that the channel is not frequency selective, so that its transfer function in the frequency domain is constant [BM, WJ, Pro].

As a result, a (component-wise) multiplicative distortion is present in the channel, as shown in the model depicted in Figure 11, where $\bar{\alpha}$ is a vector with n component i.i.d. random variables α_i, $0 \leq i < n$, each having a Rayleigh pdf,

$$p_{\alpha_i}(\alpha_i) = \alpha_i e^{-\alpha_i^2/2}, \qquad \alpha_i \geq 0. \tag{1.36}$$

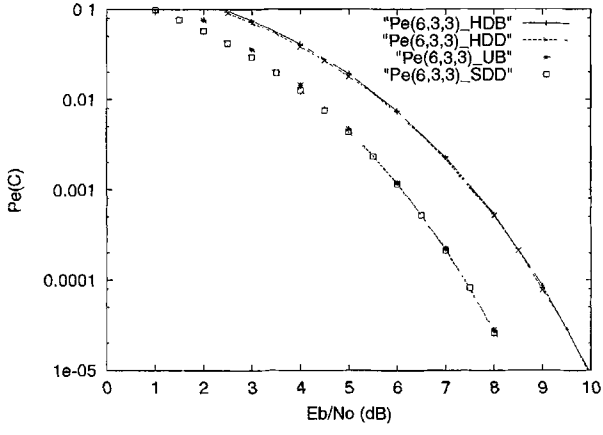


Figure 10 Simulations and union bounds for the binary (6,3,3) code. Binary transmission over an AWGN channel.

With this pdf, the average SNR per bit equals E_b/N_0 (i.e., that of the AWGN), since the second moment of the fading amplitudes is $E\{\alpha_i^2\} = 1$.

In evaluating the performance of a binary linear code over a flat Rayleigh fading channel, a conditional probability of a decoding error, $P_e(C|\bar{\alpha})$, or of a bit error, $P_b(C|\bar{\alpha})$, is computed. The unconditional error probabilities are then obtained by integration over a product of α_i, with pdf given by Equation (1.36).

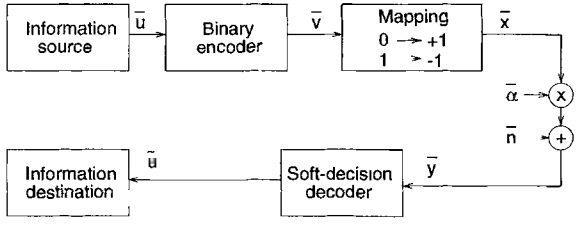


Figure 11 Binary coded transmission system over a flat Rayleigh fading channel.

The conditional probabilities of error are identical to those obtained for binary transmission over an AWGN channel. The main difference is that the arguments in the $Q(x)$ function, which correspond to the pairwise probability of a decoding error, are now weighted by the fading amplitudes α_i. Considering a coded binary transmission system without *channel state information (CSI)*, we have that

$$P_e(C|\bar{\alpha}) \leq \sum_{w=d_{min}}^{n} A_w \, Q\left(\sqrt{\frac{1}{w}\Delta_w^2 2R\frac{E_b}{N_0}}\right), \tag{1.37}$$

with

$$\Delta_w = \sum_{i=1}^{w} \alpha_i. \tag{1.38}$$

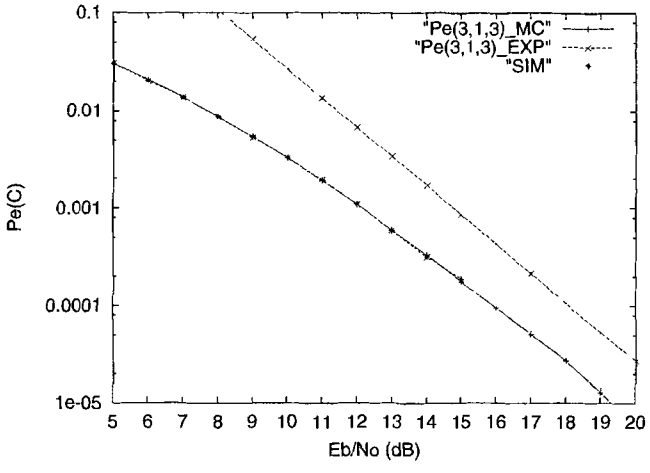


Figure 12 Simulation results (SIM), compared with the Monte Carlo union bound (Pe(3,1,3)_MC) and the Chernoff bound (Pe(3,1,3)_EXP), for the binary (3,1,3) code. Binary transmission over a flat Rayleigh fading channel.

Finally, the probability of a decoding error with binary transmission over a flat Rayleigh fading channel is obtained by taking the expectation with respect to Δ_w,

$$P_e(C) \leq \sum_{w=d_{min}}^{n} A_w \int_0^{\infty} Q\left(\sqrt{\frac{1}{w}\Delta_w^2 2R\frac{E_b}{N_0}}\right) p_{\Delta_w}(\Delta_w) d\Delta_w. \tag{1.39}$$

There are several methods to evaluate or further upper bound expression (1.39). One is to evaluate numerically (1.39) by *Monte Carlo integration*, with the approximation

$$P_e(C) \lesssim \frac{1}{N}\sum_{\ell=1}^{N}\sum_{w=d_{min}}^{n} A_w\, Q\left(\sqrt{2\Delta_w(\ell)R\frac{E_b}{N_0}}\right), \tag{1.40}$$

where $\Delta_w(\ell)$ denotes the sum of the squares of w i.i.d. random variables with Rayleigh distribution, given by (1.38), generated in the ℓ-th outcome of a computer program, and N is a sufficiently large number that depends on the range of values of $P_e(C)$. A good rule of thumb is that N should be at least 100 times larger than the inverse of $P_e(C)$. (See [Jer], pp. 500-503.)

Another method is to bound the Q-function by an exponential function (see, e.g., [WJ], pp. 82-84) and to perform the integration, or to find a *Chernoff bound*. This approach results in a loose bound that, however, yields a closed expression (see also [Wil], p. 526, and [BM], p. 718):

$$P_e(C) \leq \sum_{w=d_{min}}^{n} A_w \frac{1}{\left(1+\frac{RE_b}{N_0}\right)^w}. \tag{1.41}$$

The bound (1.41) is useful in cases where a first-order estimation of the code performance is desired.

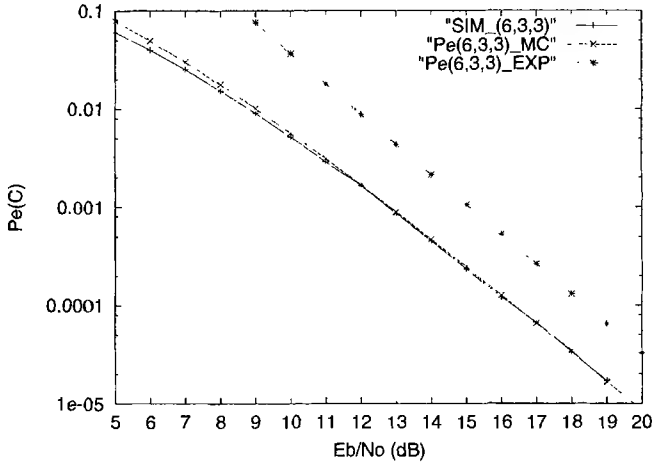


Figure 13 Simulation results (SIM_(6,3,3)), compared with the Monte Carlo union bound (Pe(6,3,3)_MC) and the Chernoff bound (Pe(6,3,3)_EXP), for the binary (6,3,3) code. Binary transmission over a flat Rayleigh fading channel.

Example 11 Figure 12 shows computer simulation results of the binary (3,1,3) code over a flat Rayleigh fading channel. Note that the Monte Carlo integration of the union bound gives exactly the actual error performance of the code, since there is only one term in the bound. Also note that the Chernoff bound is about 2 dB away from the simulations, at a signal-to-noise ratio per bit $E_n/N_0 > 18$ dB.

Example 12 Figure 13 shows the results of computer simulations of decoding the binary (6,3,3) code of Example 10, over a flat Rayleigh fading channel. In this case, the union bound is relatively loose at low values of E_b/N_0 due to the presence of more terms in the bound. Again, it can be seen that the Chernoff bound is loose by about 2 dB, at a signal-to-noise ratio per bit $E_n/N_0 > 18$ dB.

1.5 General structure of a hard-decision decoder of linear codes

In this section, the general structure of a hard-decision decoder for linear codes is summarized. Figure 14 shows a simple block diagram of the decoding process. Note that since hard decisions are assumed, bits at the output of a demodulator are fed into a decoder designed for a BSC.

Let $\bar{v} \in C$ denote a transmitted codeword. The decoder has at its input a noisy received vector $\bar{r} = \bar{v} + \bar{e}$. A two-step decoding procedure of a linear code is:

- Compute the syndrome $\bar{s} = \bar{r}H^\top$. Based on the code properties, the syndrome is a linear transformation of the error vector introduced in the channel,

$$\bar{s} = \bar{e}H^\top, \tag{1.42}$$

- Based on the syndrome $\bar{s}$, estimate the most likely error vector $\bar{e}$ and subtract it (modulo 2 addition in the binary case) from the received vector.

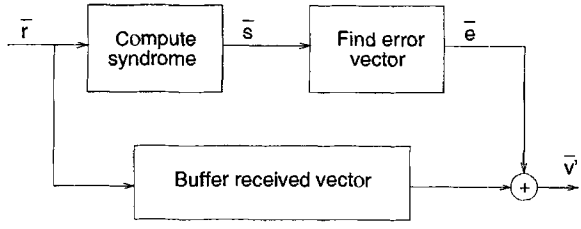


Figure 14 General structure of a hard-decision decoder of a linear block code for the BSC model.

Although most practical decoders will not perform the above procedure as stated, it is nonetheless instructive to think of a hard-decision decoder as implementing a method of solving equation (1.42). Note that any method of solving this equation constitutes a decoding method. For example, one could attempt to solve the key equation by finding a pseudo-inverse of $H^{\top}$, denoted $(H^{\top})^{+}$, such that $H^{\top}(H^{\top})^{+} = I_n$, and the solution

$$\bar{e} = \bar{s}(H^{\top})^{+}, \tag{1.43}$$

has the *smallest Hamming weight possible*. As easy as this may sound, it is a formidable task. This issue will be visited again when discussing decoding methods for BCH and Reed-Solomon codes.

2

Hamming, Golay and Reed-Muller codes

In this chapter, important examples linear binary codes are introduced. They serve to introduce more ECC concepts as well as clever decoding algorithms. Hamming codes are perhaps the most widely known class of block codes, with the possible exception of Reed-Solomon codes. As mentioned in Chapter 1, Hamming codes are optimal in the sense that they require the smallest amount of redundancy, for a given block length, to correct any single error. The binary Golay code is the only other nontrivial example of an optimal triple-error correcting code. (The only other binary optimal codes are repetition and single parity-check (SPC) codes.) Reed-Muller codes can be defined as codes with an elegant combinatorial definition that are easy to decode.

2.1 Hamming codes

Recall from Chapter 1, Equation (1.12), that any codeword $\bar{v}$ in a linear (n, k, d_{min}) block code C satisfies

$$\bar{v}H^{\top} = \bar{0}. \tag{2.1}$$

A useful interpretation of this equation is that *the maximum number of linearly independent columns of the parity-check matrix H of C is equal to $d_{min} - 1$.*

In the binary case, for $d_{min} = 3$, the above equation translates into the sum of any two columns of H not equal to the all-zero vector. Suppose that the columns of H are binary vectors of length m. There are up to $2^m - 1$ possible nonzero distinct columns. Therefore, the length of a binary single-error correcting code is given by

$$n \leq 2^m - 1.$$

This inequality is precisely the Hamming bound (1.24) for an error correcting code of length n, with $n - k = m$ and $t = 1$. Consequently, a code achieving this bound with equality is known as a *Hamming code.*

Example 13 With $m = 3$, we obtain the Hamming $(7, 4, 3)$ code, with parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

In the rest of this section, an encoding and decoding algorithm is given that is suitable for software implementation and for simulation purposes. Later in the book, effective *soft-decision* decoding algorithms are given, which can be used over real-valued channels such as the AWGN channel and the Rayleigh fading channel.

As noted before, Hamming codes have the property that their parity-check matrix H has all of its columns different. If a single error occurs, in position j, $1 \leq j \leq n$, then the syndrome of the received vector equals the column of H in the position in which the error occurred. Let $\bar{e}$ denote the error vector added in the transmission of a codeword over a BSC channel, and assume that all of its components are equal to zero except for the j-th component, $e_j = 1$. Then, the syndrome of the received word equals

$$\bar{s} = \bar{r}H^\top = \bar{e}H^\top = \bar{h}_j, \tag{2.2}$$

where $\bar{h}_j$ denotes the j-th column of H, and $1 \leq j \leq n$.

2.1.1 *Encoding and decoding procedures*

From Equation (2.2) above, it follows that if it is possible to express the columns of H as the binary representation of integers, then the value of the syndrome directly gives the position of the error. This is the idea in the algorithms for encoding and decoding presented below. The columns of the parity-check matrix H are expressed as binary representations of integer numbers i in the range $[1, n]$ and in increasing order. Let the resulting matrix be denoted by H'. Clearly, the code associated with H' is equivalent to the original Hamming code with parity-check matrix H, up to a permutation (or exchange) of positions.

Recall from Chapter 1 that the parity-check matrix in systematic form contains the $(n-k) \times (n-k)$ identity matrix, I_{n-k}, as in Equation (1.16). Clearly, when expressing H' with columns equal to the binary representation of the (integer) column number, the identity matrix is contained in those columns of H' that correspond to even powers of 2, i.e., of the form 2^ℓ, $0 \leq \ell < m$. This is illustrated in the example below.

Example 14 Let $m = 3$. Then a systematic parity-check matrix is

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix},$$

Matrix H' is given by the binary representation of integers 1 to 7 (in the following, the topmost part corresponds to the least-significant bit in the binary representation of an integer):

$$H' = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

and matrix I_{n-k} is contained in columns 1, 2 and 4.

In general, for a $(2^m - 1, 2^m - 1 - m)$ Hamming code, the identity matrix is contained in column numbers $1, 2, 4, \cdots, 2^{m-1}$ of H'.

Encoding

Encoding proceeds as dictated by Equation (1.18) of Chapter 1. When computing the parity check bit p_j, for $1 \leq j \leq m$, the column position numbers are examined and those which are not powers of two correspond to message positions and the corresponding message bits are included in the computation. This encoding procedure is somewhat more complicated than that of a systematic Hamming code. In return, however, decoding is extremely simple. Depending on the application, this approach may prove the most appropriate, since it is usually decoding that is required to be very fast.

Decoding

Once the codewords have been computed in accordance to matrix H', decoding is easy. The syndrome (2.2) equals the position number in which the error occurred! In a decoding procedure, after the computation of syndrome s, when regarded as an integer, the erroneous position is then corrected as

$$v_s = v_s \oplus 1,$$

where $\oplus$ denotes exclusive-or (i.e., $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, and $1 \oplus 1 = 0$).

Program `hamming.c` in the ECC home page implements the above encoding and decoding procedures for binary Hamming codes.

2.2 The binary Golay code

Golay [Gol] noticed that

$$\sum_{i=0}^{3} \binom{23}{i} = 2^{11}.$$

This equality shows the possible existence of a perfect binary $(23, 12, 7)$ code, with $t = 3$, that is, capable of correcting all possible patterns of at most three errors in 23 bit positions. In his paper, Golay gave a generator matrix of such a triple-error correcting binary code.

Because of its relatively small length (23), dimension (12) and number of redundant bits (11), the binary $(23, 12, 7)$ Golay code can be encoded and decoded simply by using look-up tables. The program `golay23.c` in the ECC home page uses a 16K $\times$ 23 bits encoding table and an 8K $\times$ 23 bits decoding table.

2.2.1 Encoding

Encoding is based on a look-up table (LUT) that contains a list of all the $2^{12} = 4096$ codewords, which are indexed directly by the data. Let $\bar{u}$ denote a 12-bit binary vector representing the data to be encoded, and let $\bar{v}$ denote the corresponding 23-bit codeword. The encoder LUT is constructed by generating all 4096 12-bit vectors and computing the syndrome of a pattern for which the 12 MSB equal to the information bits and the 11 LSB equal to zero. The 11-bit syndrome then becomes the LSB part of the codeword.

The LUT is a one-to-one mapping from $\bar{u}$ onto $\bar{v}$, which can be expressed as

$$\bar{v} = \text{LUT}(\bar{u}) = (\bar{u}, \text{get\_syndrome}(\bar{u}, \bar{0})). \tag{2.3}$$

In the construction of the encoder LUT, advantage is taken from the cyclic nature of the Golay code. Its *generator polynomial*[1] is

$$g(x) = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1,$$

which in *hexadecimal notation* equals `C75`. This polynomial is used to generate the syndrome in the procedure "get_syndrome" indicated in Equation (2.3) above.

2.2.2 *Decoding*

Recall from Chapter 1, Figure 14, that the decoder's task is to estimate the most-likely (i.e., least Hamming weight) error vector $\bar{e}$ from the received vector $\bar{r}$. The decoder for the Golay code is based on an LUT that accepts as input the syndrome $\bar{s}$ of the received vector $\bar{r}$ and outputs the error vector $\bar{e}$.
The procedure to construct the decoder LUT is as follows:

1. Generate all possible error patterns $\bar{e}$ of Hamming weight less than or equal to three;
2. For each error pattern, compute the corresponding syndrome $\bar{s}$ = get_syndrome($\bar{e}$);
3. Store at location $\bar{s}$ of the LUT, the error vector $\bar{e}$,

$$\text{LUT}(\bar{s}) = \bar{e}.$$

With the decoder LUT, upon reception of a corrupted received word $\bar{r}$, correction of up to three bit errors is accomplished by the following:

$$\hat{v} = \bar{r} \oplus \text{LUT}\left(\text{get\_syndrome}(\bar{r})\right),$$

where $\hat{v}$ denotes the corrected word.

2.2.3 *Arithmetic decoding of the extended (24, 12, 8) Golay code.*

In this section, a decoding procedure for the extended $(24, 12, 8)$ Golay code, C_{24}, is presented based on an arithmetic decoding algorithm [Wic, VO]. The algorithm utilizes the rows and columns of the parity submatrix B in the parity-check matrix $H = (\, B \mid I_{12\times 12}\,)$. Note that an extended $(24, 12, 8)$ Golay code, C'_{24}, which is equivalent to C_{24} up to a permutation in the bit positions, can be obtained by adding an overall parity-check bit at the end of each codeword in the $(23, 12, 7)$ Golay code.

In hexadecimal notation, the twelve rows of B, denoted row_i, $1 \le i \le 12$, are:

```
0x7ff, 0xee2, 0xdc5, 0xb8b, 0xf16, 0xe2d,
0xc5b, 0x8b7, 0x96e, 0xadc, 0xdb8, 0xb71.
```

It is interesting to note that sub-matrix B of the parity-check matrix of C_{24} satisfies $B = B^{\top}$. This means that code C_{24} is a *self-dual code*. Details of self-dual codes are not covered in this book. Interested readers are referred to [MS, Wic].

In program `golay24.c`, encoding is performed by recurrence with H, as indicated in Equation 1.18. As before, let $\text{wt}_H(\bar{x})$ denote the Hamming weight of a vector $\bar{x}$. Decoding steps are as follows [Wic, VO]:

---

[1] The definition of a generator polynomial is given in Chapter 3.

1. Compute the syndrome $\bar{s} = \bar{r}H^\top$.
2. If $\mathrm{wt}_H(\bar{s}) \leq 3$, then set $\bar{e} = (\bar{s}, \bar{0})$ and go to step 8.
3. If $\mathrm{wt}_H(\bar{s} + \mathrm{row}_i) \leq 2$, then set $\bar{e} = (\bar{s} + \mathrm{row}_i, \bar{x}_i)$, where $\bar{x}_i$ is a 12-bit vector with only the i-th coordinate nonzero.
4. Compute $\bar{s}B$.
5. If $\mathrm{wt}_H(\bar{s}B) \leq 3$, then set $\bar{e} = (\bar{0}, \bar{s}B)$ and go to step 8.
6. If $\mathrm{wt}_H(\bar{s}B + \mathrm{row}_i) \leq 2$, then set $\bar{e} = (\bar{x}_i, \bar{s}B + \mathrm{row}_i)$, with $\bar{x}_i$ defined as above, and go to step 8.
7. $\bar{r}$ is corrupted by an uncorrectable error pattern, set error failure flag. End of decoding.
8. Set $\hat{c} = \bar{r} + \bar{e}$. End of decoding.

2.3 Binary Reed-Muller codes

Binary Reed-Muller (RM) codes constitute a family of error correcting codes that are easy to decode using *majority-logic circuits*. In addition, codes in this family are known to have relatively simple and highly structured trellises [LKFF]. More on trellises of linear block codes is discussed in Chapter 7.

An elegant definition of binary RM code is obtained with the use of binary polynomials (or Boolean functions). With this definition, RM codes become close relatives of BCH codes and RS codes, all members of the class of *polynomial codes*[2].

2.3.1 *Boolean polynomials and RM codes*

This section closely follows the development of [MS]. Let $f(x_1, x_2, \cdots, x_m)$ denote a *Boolean function* on m binary-valued variables $x_1, x_2, \cdots, x_m$. It is well known that such a function can be specified by a *truth table*. The truth table lists the value of f for all 2^m combinations of values of its arguments. All the usual Boolean operations (such as "AND","OR") can be defined in a Boolean function.

Example 15 Consider the function $f(x_1, x_2)$ with the following truth table:

x_2	0	0	1	1
x_1	0	1	0	1
$f(x_1, x_2)$	0	1	1	0

Then,

$$f(x_1, x_2) = (x_1 \text{ AND NOT}(x_2)) \text{ OR } (\text{NOT}(x_1) \text{ AND } x_2)\,.$$

Associated with each Boolean function f, let $\bar{f}$ denote the binary vector of length 2^m which is obtained from evaluating f at all possible 2^m values of the m variables $x_1, x_2, \cdots, x_m$. In the example above, $\bar{f} = (0110)$, where the convention taken for ordering the bit positions of $\bar{f}$ is in accordance to a binary representation of integers, with x_1 being the least-significant bit (LSB) and x_m the most-significant bit (MSB).

Note also that a Boolean function can be written directly from its truth table to get the *disjunctive normal form* (DNF). Using the DNF, any Boolean function can be expressed as

[2] Polynomial codes are presented in Section 3.4.

the sum[3] of 2^m elementary functions: $1, x_1, x_2, \cdots, x_m, x_1x_2, \cdots, x_1x_2\cdots x_m$, such that

$$\bar{f} = \bar{1} + a_1\bar{x}_1 + a_2\bar{x}_2 + \cdots + a_m\bar{x}_m + a_{12}\bar{x}_1\bar{x}_2 + \cdots + a_{12\cdots m}\bar{x}_1\bar{x}_2\cdots\bar{x}_m, \qquad (2.4)$$

where $\bar{1}$ is added to account for independent terms (degree 0). For Example 15, $\bar{f} = \bar{x}_1 + \bar{x}_2$.

A binary RM $(2^m, k, 2^{m-r})$ code, denoted $\mathrm{RM}_{r,m}$, is defined as the set of vectors associated with all Boolean functions of degree up to r in m variables. $\mathrm{RM}_{r,m}$ is also known as the r-th order RM code of length 2^m. The dimension of $\mathrm{RM}_{r,m}$ can easily by shown to be equal to

$$k = \sum_{i=0}^{r} \binom{m}{i},$$

which corresponds to the number of ways polynomials of degree up to r can be constructed with m variables.

In view of Equation (2.4), a generator matrix for $\mathrm{RM}_{r,m}$ is formed by taking as rows the vectors associated with the k Boolean functions which can be expressed as polynomials of degree up to r in m variables.

Example 16 The first-order RM code of length 8, $\mathrm{RM}_{1,3}$, is an $(8, 4, 4)$ binary code, and can be constructed from Boolean functions of degree up to 1 in 3 variables: $\{1, x_1, x_2, x_3\}$, so that

$$\begin{array}{lcccccccc} \bar{1} = & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \bar{x}_1 = & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \bar{x}_2 = & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ \bar{x}_3 = & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

A generator matrix for $\mathrm{RM}_{1,3}$ is thus

$$G = \begin{pmatrix} \bar{1} \\ \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \qquad (2.5)$$

Note that code $\mathrm{RM}_{1,3}$ can be also obtained from a Hamming $(7, 4, 3)$ code by appending at the end of each codeword an overall parity-check bit. The only difference between the extended Hamming code and $\mathrm{RM}_{1,3}$ will be a possible permutation of bit (column) positions.

Dual codes of RM codes are also RM codes

It can be shown that $\mathrm{RM}_{m-r-1,m}$ is the dual code of $\mathrm{RM}_{r,m}$. In other words, the generator matrix of $\mathrm{RM}_{m-r-1,m}$ can be used as a parity-check matrix of $\mathrm{RM}_{r,m}$.

2.3.2 *Finite geometries and majority-logic decoding*

RM codes can be formulated in terms of a *finite geometry*. An *Euclidean geometry* $EG(m, 2)$ of dimension m over $GF(2)$ consists of 2^m *points*, which are all the binary vectors of length

[3] "sum" means logical "XOR" and "multiplication" means logical "AND" in this context.

m. Note that the columns of the matrix formed by the last three rows of the generator matrix of $\mathrm{RM}_{1,3}$, see Example 16 above, are the 8 points of $EG(3,2)$. If the zero point is deleted, then a *projective geometry* $PG(m-1,2)$ is obtained. The reader is referred to [LC] for an excellent treatment of finite geometries and RM codes. Finite geometry codes are in fact a generalization of RM codes[4].

The connection between codes and finite geometries can be introduced as follows: Consider $EG(m,2)$. The columns of the matrix $(\bar{x}_1^\top \bar{x}_2^\top \cdots \bar{x}_m^\top)^\top$ are taken as the coordinates of points of the geometry $EG(m,2)$. Then, there is a one-to-one correspondence between the components of binary vectors of length 2^m and the points of $EG(m,2)$. A given binary vector of length 2^m is associated with a subset of points of $EG(m,2)$. In particular, a subset of $EG(m,2)$ can be associated with each binary vector $\bar{w} = (w_1, w_2, \cdots, w_{2^m})$ of length 2^m, by interpreting it as selecting points whenever $w_i = 1$. Stated otherwise, $\bar{w}$ is an *incidence vector.*

Binary Reed-Muller codes can then be defined as follows: The codewords of $\mathrm{RM}_{r,m}$ are the incidence vectors of all subspaces (i.e., linear combinations of points) of dimension $m-r$ in $EG(m,2)$ (Theorem 8 of [MS]). From this it follows that the number of minimum weight codewords of $\mathrm{RM}_{r,m}$ is

$$A_{2^{m-r}} = 2^r \prod_{i=0}^{m-r-1} \frac{2^{m-i}-1}{2^{m-r-i}-1}. \tag{2.6}$$

The code obtained by deleting (or puncturing) the coordinate corresponding to $x_1 = x_2 = \cdots = x_m = 0$ from all the codewords of $\mathrm{RM}_{r,m}$ is the binary *cyclic* $\mathrm{RM}^\star_{r,m}$ code, which as

$$A^\star_{2^{m-r}-1} = \prod_{i=0}^{m-r-1} \frac{2^{m-i}-1}{2^{m-r-i}-1}. \tag{2.7}$$

minimum-weight codewords.

In terms of decoding, it turns out that RM codes can be decoded with so-called *majority-logic (ML) decoding.* The idea is the following: The parity-check matrix induces 2^{n-k} parity-check equations. Designing an ML decoder is selecting a subset of the parity-check equations in such a way that a majority vote is taken on the value of a specific code position. As an illustration, consider again the $\mathrm{RM}_{1,3}$ code of Example 16 above.

Example 17 Let $\bar{v} = \bar{u}G = (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8)$ be a codeword in $\mathrm{RM}_{1,3}$. As mentioned before, in this case equation (2.5) is also a parity-check matrix, since $r = 1$ and $m-r-1 = 1$, and the code is *self-dual.* All possible 15 nonzero combinations of parity-check equations (rows of H) are the following:

$$\begin{aligned}
v_1 + v_2 + v_3 + v_4 + v_5 + v_6 + v_7 + v_8 &= 0 \\
v_5 + v_6 + v_7 + v_8 &= 0 \\
v_3 + v_4 + v_7 + v_8 &= 0 \\
v_2 + v_4 + v_6 + v_8 &= 0 \\
v_1 + v_2 + v_3 + v_4 &= 0 \\
v_1 + v_2 + v_5 + v_6 &= 0
\end{aligned}$$

[4] See Section 3.4.

$$\begin{aligned} v_1 + v_3 + v_5 + v_7 &= 0 \\ v_3 + v_4 + v_5 + v_6 &= 0 \\ v_2 + v_4 + v_5 + v_7 &= 0 \\ v_2 + v_3 + v_6 + v_7 &= 0 \\ v_1 + v_2 + v_7 + v_8 &= 0 \\ v_1 + v_3 + v_6 + v_8 &= 0 \\ v_1 + v_4 + v_5 + v_8 &= 0 \\ v_2 + v_3 + v_5 + v_8 &= 0 \\ v_1 + v_4 + v_6 + v_7 &= 0 \end{aligned} \tag{2.8}$$

The reader is invited to verify that the sum $v_i + v_j$ of every pair of code bits v_i, v_j, with $i \neq j$, appears in exactly four equations. Whenever a set of equations includes a term $v_i + v_j$, but no other sum of pairs appears more than once, the parity-checks involved are said to be *orthogonal* on positions v_i and v_j.

It is now shown how a single error can be corrected. Let

$$\bar{r} = \bar{v} + \bar{e} = (r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8)$$

denote a received vector after codeword $\bar{v}$ is transmitted over a BSC. Suppose that a single error is to be corrected in the fifth position, v_5. A procedure to design an ML decoder for this case is as follows:

Two equations involving the term $v_i + v_5$ are selected, with $i \neq 5$, and another set of two equations with the term $v_j + v_5$, $j \neq 5$, with $i \neq j$. Select (arbitrarily, as long as $i \neq j$ and both different than 5), say, $i = 3, j = 4$. There are four parity checks orthogonal to the term $v_3 + v_5$. Select any two of them. Do the same for the term $v_4 + v_5$.

The *syndromes* associated with these equations are denoted S_1 and S_2 for $v_3 + v_5$ and S_3 and S_4 for $v_4 + v_5$,

$$\begin{aligned} S_1 &\triangleq r_1 + \mathbf{r_3} + \mathbf{r_5} + r_7 \\ S_2 &\triangleq \mathbf{r_3} + r_4 + \mathbf{r_5} + r_6 \\ S_3 &\triangleq r_2 + \mathbf{r_4} + \mathbf{r_5} + r_7 \\ S_4 &\triangleq r_1 + \mathbf{r_4} + \mathbf{r_5} + r_8 \end{aligned} \tag{2.9}$$

Because $\bar{v}$ is a codeword in $\mathrm{RM}_{1,3}$, the set of equations (2.9) is equivalent to

$$\begin{aligned} S_1 &\triangleq e_1 + \mathbf{e_3} + \mathbf{e_5} + e_7 \\ S_2 &\triangleq \mathbf{e_3} + e_4 + \mathbf{e_5} + e_6 \\ S_3 &\triangleq e_2 + \mathbf{e_4} + \mathbf{e_5} + e_7 \\ S_4 &\triangleq e_1 + \mathbf{e_4} + \mathbf{e_5} + e_8 \end{aligned} \tag{2.10}$$

Since S_1, S_2, S_3 and S_4 are orthogonal on $e_3 + e_5$ and $e_4 + e_5$, a new pair of equations orthogonal on e_5 can be formed as:

$$\begin{aligned} S_1' &\triangleq e_3' + e_5' \\ S_2' &\triangleq e_4' + e_5' \end{aligned} \tag{2.11}$$

where e'_j, $j = 3, 4, 5$, represents the ML estimate from (2.9). Equations (2.11) are orthogonal on e'_5 and consequently the value of e'_5 can be obtained by a majority vote, for example, setting it to the output of an "AND" gate with inputs S'_1 and S'_2.

Suppose that the codeword $\bar{v} = (11110000)$ is transmitted and $\bar{r} = (11111000)$ is received. Then (2.10) gives:

$$\begin{aligned} S_1 &= r_1 + \mathbf{r_3} + \mathbf{r_5} + r_7 = 1 \\ S_2 &= \mathbf{r_3} + r_4 + \mathbf{r_5} + r_6 = 1 \\ S_3 &= r_2 + \mathbf{r_4} + \mathbf{r_5} + r_7 = 1 \\ S_4 &= r_1 + \mathbf{r_4} + \mathbf{r_5} + r_8 = 1 \end{aligned} \tag{2.12}$$

Thus both $e_3 + e_5$ $e_4 + e_5$ are estimated as having value equal to '1'. From (2.11), it is concluded that $e_5 = 1$, and the estimated error vector is $\bar{e} = (00001000)$, from which the estimated codeword is $\bar{v} = \bar{r} + \bar{e} = (11110000)$. This shows how an error in the fifth position can be corrected with a *two-step* ML decoder.

In the previous example, it was shown how an error in a specific position of an $\mathrm{RM}_{1,3}$ code can be corrected. A similar procedure can be applied to every position in the code. Therefore, a total of eight ML estimates will be obtained.

In general, an $\mathrm{RM}_{r,m}$ code can be decoded with an $(r+1)$-step ML decoder capable of correcting any combination of up to $\lfloor (2^{m-2} - 1)/2 \rfloor$ random errors [MS, LC].

In addition, a cyclic $\mathrm{RM}^{\star}_{r,m}$ code is simpler to decode. In a cyclic code[5] C, if $(v_1, v_2, \cdots, v_n)$ is a codeword of C, then the *cyclic shift* $(v_n, v_1, \cdots, v_{n-1})$ also is a codeword of C. Therefore, once a position can be corrected via ML decoding, the remaining positions can also be corrected with the same algorithm (or hardware circuit) by cyclically shifting received codewords, until all n positions have been tried.

Example 18 In this example, a decoder for the cyclic $\mathrm{RM}^{\star}_{1,3}$ code, a binary Hamming $(7, 4, 3)$ code, is derived. To obtain the parity-check equations from those of the $\mathrm{RM}_{1,3}$ code, remove the coordinate v_1 for which $x_1 = x_2 = \cdots = x_m$ from all equations. Let the codewords of $\mathrm{RM}^{\star}_{1,3}$ be indexed by re-labeling the codeword elements: $(v_2, v_3, v_4, v_5, v_6, v_7, v_8) \rightarrow (v_1, v_2, v_3, v_4, v_5, v_6, v_7)$. As before, an ML decoder for correcting an error in an arbitrary position (say, the *fifth* position again) can be derived. This can be shown to result in the following seven nonzero (linearly independent) parity-check equations:

$$\begin{aligned} v_1 + v_2 + v_3 + v_5 &= 0 \\ v_2 + v_3 + v_4 + v_6 &= 0 \\ v_3 + v_4 + v_5 + v_7 &= 0 \\ v_1 + v_4 + v_5 + v_6 &= 0 \\ v_2 + v_5 + v_6 + v_7 &= 0 \\ v_1 + v_3 + v_6 + v_7 &= 0 \\ v_1 + v_2 + v_4 + v_7 &= 0 \end{aligned} \tag{2.13}$$

[5] See Chapter 3.

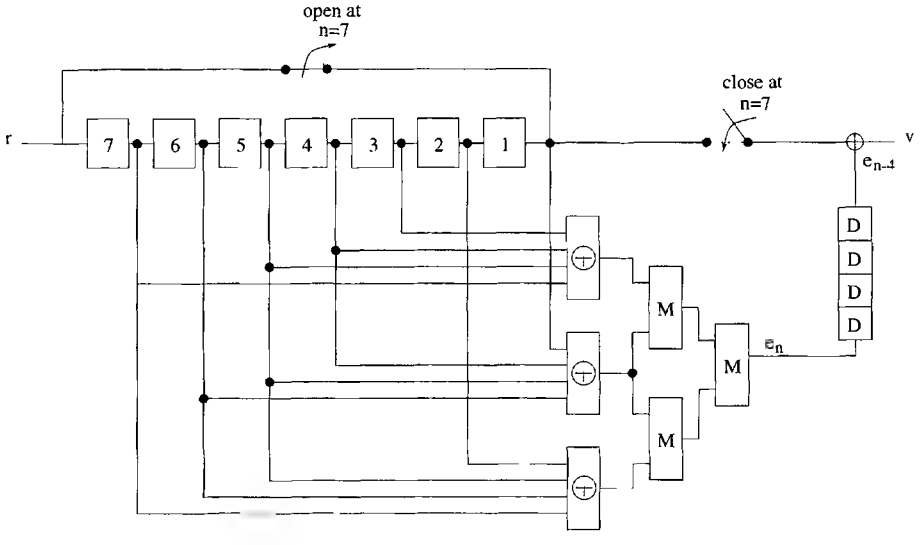


Figure 15 A majority-logic decoder for a cyclic RM*(1, 3) code.

In a manner similar to the previous example, the syndromes S_1 and S_2 below are orthogonal on v_4 and v_5, and S_2 and S_3 are orthogonal on v_5 and v_6:

$$\begin{aligned} S_1 &= e_3 + e_4 + e_5 + e_7 \\ S_2 &= e_1 + e_4 + e_5 + e_6 \\ S_3 &= e_2 + e_5 + e_6 + e_7 \end{aligned} \tag{2.14}$$

Based on the estimates $e'_4 + e'_5$ and $e'_5 + e'_6$ two additional orthogonal equations on e_5 can be formed to give the final estimate,

$$\begin{aligned} S'_1 &= e'_4 + e'_5 \\ S'_2 &= e'_5 + e'_6 \end{aligned} \tag{2.15}$$

where e'_j, $j = 4, 5, 6$, represents the ML estimate from the previous step. This results in the circuit shown in the Figure 15. The circuit operates as follows. Initially the contents of the seven-bit register are set to zero. Suppose that a single error is contained in the received word in position i, for $1 \leq i \leq 7$. At each clock cycle, the contents of the register are cyclically shifted to the right by one position. Time, in clock cycles, is denoted by the subindex n in the following.

Consider first the case $i = 1$. That is, there is an error in the first codeword position. After three cycles, the error is contained in register 5 (v_5). The output of the majority-logic circuit is set to $e_n = 1$. Four cycles later (a total of seven cycles), the first received bit is output and the error corrected. Consider now the case $i = 7$. After nine cycles, the error is detected and $e_n = 1$. Again, four cycles later (total 13 cycles), the bit in the last position is output and the error corrected. This decoder has a latency of 13 cycles. Every 13 cycles, the contents of the shift register are cleared and a new codeword can be processed.

In the next chapter, binary cyclic codes and the powerful family of BCH codes are introduced.

3

Binary cyclic codes and BCH codes

The aim of this chapter is to introduce a minimum set of concepts necessary for the understanding of binary cyclic codes and for the efficient implementation of their encoding and decoding procedures. Also in this chapter, the important family of binary BCH codes is introduced. BCH codes are a family of *cyclic codes*, which gives them an algebraic structure that is useful in simplifying their encoding and decoding procedures. Binary BCH codes with minimum distance 3, better known as *Hamming codes*, have been a very popular choice in computer networks and in memories, due to their simple and fast encoding and decoding. Also, a shortened $(48, 36, 5)$ BCH code is used in the U.S. cellular TDMA system specification, standard IS-54.

3.1 Binary cyclic codes

Cyclic codes are a class of error correcting codes that can be efficiently encoded and decoded using simple shift-registers and combinatorial logic elements, based on their representation using polynomials. In this section, the fundamental concepts of cyclic codes are discussed.

3.1.1 Generator and parity-check polynomials

Let C denote a linear (n, k) block code. Let $\bar{u}$ and v denote a *message* and the corresponding *codeword* in C, respectively. Cyclic codes are linear codes with properties that make them suitable for hardware implementation. To every codeword $\bar{v}$ *a polynomial* $\bar{v}(x)$ is associated,

$$\bar{v} = (v_0, v_1, \cdots, v_{n-1}) \mapsto \bar{v}(x) = v_0 + v_1 x + \cdots v_{n-1} x^{n-1}.$$

The indeterminant x serves to indicate the relative position of an element v_i of $\bar{v}$ as a term $v_i x^i$ of $\bar{v}(x)$, $0 \leq i < n$.

A linear block code C is cyclic if and only if *every cyclic shift of a codeword is another codeword*,

$$\bar{v} = (v_0, v_1, \cdots, v_{n-1}) \in C \iff \bar{v}^{(1)} = (v_{n-1}, v_0, \cdots, v_{n-2}) \in C.$$

In the language of polynomials, a cyclic shift by one position, denoted $\bar{v}^{(1)}(x)$, is accomplished by a multiplication by x modulo $(x^n - 1)$,

$$\bar{v}(x) \in C \iff \bar{v}^{(1)}(x) = x\bar{v}(x) \bmod (x^n - 1) \in C.$$

Shift-registers can be used for this purpose, as illustrated in Figure 16.

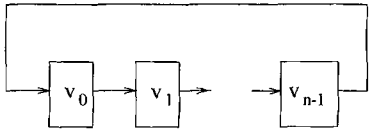


Figure 16 A cyclic shift register

Example 19 Consider the case of $n = 7$. Then, a cyclic shift by one position of the vector $\bar{v} = (0101011)$ equals $\bar{v}^{(1)} = (1010101)$. In terms of polynomials, $\bar{v}(x) = x + x^3 + x^5 + x^6$ and

$$\begin{aligned} \bar{v}^{(1)}(x) = x\bar{v}(x) &= x^2 + x^4 + x^6 + x^7 \bmod (x^7 + 1) \\ &= x^2 + x^4 + x^6 + x^7 + (x^7 + 1) + 1 \\ &= 1 + x^2 + x^4 + x^6 \end{aligned}$$

3.1.2 The generator polynomial

An important property of cyclic codes is that all code polynomials $\bar{v}(x)$ are multiples of a unique polynomial, $\bar{g}(x)$, called the *generator polynomial* of the code. This polynomial is *specified by its roots*, which are called the *zeros of the code*. It can be shown that the generator polynomial $g(x)$ divides $(x^n - 1)$. (As with integers, "$a(x)$ divides $b(x)$" whenever $b(x) = q(x)a(x)$.) Therefore, to find a generator polynomial, the polynomial $(x^n - 1)$ must be factored into its *irreducible factors*, $\phi_j(x)$, $j = 1, 2, \cdots, \ell$,

$$(x^n - 1) = \phi_1(x)\phi_2(x) \cdots \phi_\ell(x). \tag{3.1}$$

Also, note that over the field of binary numbers, $a - b$ and $a + b$ (modulo 2) give the same result. In the remainder of the text, as all the codes are defined over the binary field, or its extensions, no distinction is made between these operations.

As a consequence of the above, the polynomial $\bar{g}(x)$ is given by

$$\bar{g}(x) = \prod_{j \in J \subset \{1,2,\cdots,\ell\}} \phi_j(x). \tag{3.2}$$

Example 20 With coefficients over $Z_2 = \{0, 1\}$, the polynomial $x^7 + 1$ factors as

$$x^7 + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Some examples of cyclic codes of length 7 are:

- A binary cyclic Hamming $(7, 4, 3)$ code is generated by the polynomial

$$\bar{g}(x) = x^3 + x + 1.$$

- A binary cyclic parity-check $(7, 6, 2)$ code is generated by

$$\bar{g}(x) = (x^3 + x + 1)(x^3 + x^2 + 1).$$

- A binary *maximum-length-sequence* $(7, 3, 4)$ code is generated by

$$\bar{g}(x) = (x + 1)(x^3 + x + 1).$$

3.1.3 *Encoding and decoding of binary cyclic codes*

The dimension of a binary cyclic (n, k) code is given by

$$k = n - \deg\,[\bar{g}(x)],$$

where deg $[\cdot]$ denotes the degree of the argument. Since a cyclic code C is also linear, any set of k linearly independent (LI) vectors can be selected as a generator matrix. In particular, the binary vectors associated with $\bar{g}(x)$, $x\bar{g}(x)$, $\cdots$, $x^{k-1}\bar{g}(x)$ are LI. These vectors can be used as rows of a generator matrix of C. In this case a *non-systematic* encoding rule is achieved. That is, the message bits do not appear explicitly in any positions of the codewords.

Example 21 Consider the cyclic Hamming $(7, 4, 3)$ code, with generator polynomial $\bar{g}(x) = x^3 + x + 1 \iff \bar{g} = (1101)$. A generator matrix for this code is

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

Alternatively, the parity sub-matrix of a generator matrix of a cyclic code can be constructed with the vectors associated with the following polynomials:

$$\begin{gathered} x^{n-1} \bmod \bar{g}(x), \\ \vdots \\ x^{n-k+1} \bmod \bar{g}(x), \\ x^{n-k} \bmod \bar{g}(x), \end{gathered}$$

and a *systematic* encoding is obtained, as illustrated in the example below.

Example 22 Let C be the cyclic Hamming $(7, 4, 3)$ code. Then $\bar{g}(x) = x^3 + x + 1$, and

$$\begin{aligned} x^6 \bmod (x^3 + x + 1) &= x^2 + 1, \\ x^5 \bmod (x^3 + x + 1) &= x^2 + x + 1, \\ x^4 \bmod (x^3 + x + 1) &= x^2 + x, \\ x^3 \bmod (x^3 + x + 1) &= x + 1. \end{aligned}$$

It follows that a systematic generator matrix of C is

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Let $\bar{u}(x)$ be associated with a message to be encoded. Encoding of codewords of a binary cyclic code can be either non-systematic or systematic, depending on the way that the message is processed:

- Non-systematic encoding

$$\bar{v}(x) = \bar{u}(x)\bar{g}(x). \tag{3.3}$$

- Systematic encoding

$$\bar{v}(x) = x^{n-k}\bar{u}(x) + [x^{n-k}\bar{u}(x) \bmod \bar{g}(x)]. \tag{3.4}$$

3.1.4 *The parity-check polynomial*

Another polynomial, $\bar{h}(x)$, called the *parity-check* polynomial, can be associated with the parity-check matrix. The generator polynomial and the parity-check polynomial are related by

$$\bar{g}(x)\bar{h}(x) = x^n + 1. \tag{3.5}$$

The parity-check polynomial can be computed from the generator polynomial as $\bar{h}(x) = (x^n + 1)/\bar{g}(x) = h_0 + h_1 x + \cdots + h_k x^k$. Then, a parity-check matrix for C is given by using as rows the binary vectors associated with the first $n - k - 1$ nonzero cyclic shifts $\bar{h}^{(j)}(x) = x^j \bar{h}(x) \bmod (x^n - 1)$, $j = 0, 1, \cdots, n - k - 1$.

$$H = \begin{pmatrix} h_0 & h_1 & \cdots & & h_k & 0 & 0 & \cdots & 0 \\ 0 & h_0 & h_1 & \cdots & \cdots & h_k & 0 & \cdots & 0 \\ 0 & 0 & h_0 & h_1 & \cdots & \cdots & h_k & \cdots & 0 \\ 0 & 0 & 0 & \ddots & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots & h_k \end{pmatrix}. \tag{3.6}$$

Example 23 The parity-check polynomial for the cyclic Hamming (7,4,3) code, with generator polynomial $\bar{g}(x) = x^3 + x + 1$, is $\bar{h}(x) = (x^7 + 1)/(x^3 + x + 1) = x^4 + x^2 + x + 1$. A parity-check matrix for this code is

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

In the same manner as with linear codes, systematic encoding can be also performed by a recursion with $h(x)$ using

$$\bar{v}(x)\bar{h}(x) = 0 \bmod (x^n + 1).$$

Systematic encoding can be performed as follows [PW, LC]. Suppose first that the code rate $k/n \leq 0.5$. Let $\bar{u}(x) = u_0 + u_1 x + \cdots + u_{k-1}$ denote a polynomial of degree $k - 1$, whose coefficients u_ℓ, $\ell = 0, 1, \cdots, k - 1$, are the message bits to be encoded. Let $\bar{v}(x)$ denote the code polynomial polynomial in C corresponding to the information polynomial $\bar{u}(x)$. In the first step, $v_\ell = u_\ell$, for $\ell = 0, 1, \cdots, k - 1$.

From the cyclic structure of the code, it follows that the redundant bits v_ℓ, $\ell = k, k + 1, \cdots, n - 1$, are obtained recursively via the parity-check relations

$$v_\ell = \sum_{j=0}^{\ell-1} v_j h_{(\ell-k),j}, \quad \ell = k, k + 1, \cdots, n - 1 \tag{3.7}$$

where $h_{(\ell-k),j}$ denotes the j-th entry in the $(\ell - k)$-th row of matrix (3.6).

In the case of high-rate cyclic (n, k) codes, say $k/n > 0.5$, encoding by division of $x^{n-k}\bar{u}(x)$ by $\bar{g}(x)$ is more efficient. Either way, by recursion with $\bar{h}(x)$ or by division by $\bar{g}(x)$, the coefficients of the code polynomials are in systematic form, so that the first k coefficients are the message bits, and the remaining $n - k$ coefficients constitute the redundant bits.

Figure 17 shows the block diagram of an encoder for a binary cyclic code with generator polynomial $\bar{g}(x)$. Initially, the switch (lower right portion of the figure) is in position 1, and the message bits are both transmitted through the channel and simultaneously fed to a sequential

circuit that multiplies by x^{n-k} and divides by $\bar{g}(x)$. After k cycles, the shift register contains the remainder of the division, the switch moves to position 2 and the contents of the register are sent through the channel.

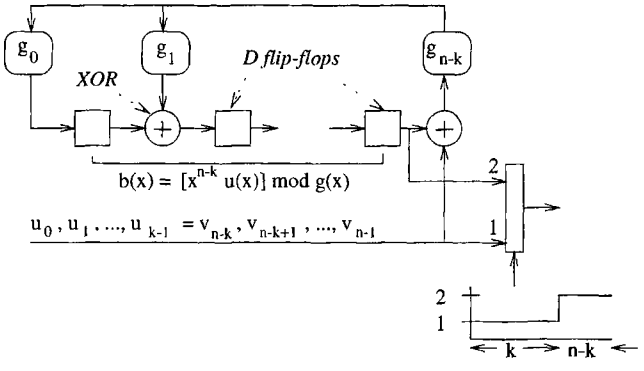


Figure 17 Circuit for systematic encoding: Division by $\bar{g}(x)$.

Duals of cyclic codes and maximum-length-sequence codes

By analogy with linear codes, the *dual code* of a cyclic code C with generator polynomial $\bar{g}(x)$ is the cyclic code $C^{\perp}$ generated by the polynomial $\bar{h}(x)$. The important class of *maximum-length-sequence* (MLS) cyclic codes [PW] has as members the duals of the cyclic Hamming codes. An MLS cyclic $(2^m - 1, m, 2^{m-1})$ code is generated by the polynomial $\bar{g}(x) = (x^n + 1)/p(x)$, where $p(x)$ is a *primitive polynomial*[1].

3.1.5 Shortened cyclic codes and CRC codes

There are many practical applications in which an error correcting code with simple encoding and decoding procedures is desired, but existing constructions do not give the desired length, dimension and minimum distance.

The following is an example from an email recently sent to the author: *We plan to use a simple FEC/ECC scheme to detect/correct single-bit errors in a 64-bit data block. The objective is to find or choose an ECC scheme to correct single-bit errors with up to 8 bits of overhead, giving a maximum of 72 bits (64 data bits plus 8 redundant bits) in total.*

Naturally, since 72 is not of the form $2^m - 1$, none of the cyclic codes studied so far can be applied directly. One possible solution is to use a cyclic Hamming $(127, 120, 3)$ code and to *shorten* it until a dimension $k = 64$ is reached. This yields a *shortened* Hamming $(71, 64, 3)$ code[2].

Shortening is accomplished by not using all the information bits of a code. Let s

[1] For a definition of primitive polynomial, see page 41.

[2] This is an example to introduce the concept of shortening. A (72,64,4) single-error-correcting/double-error-detecting (SEC/DED) code, based on a shortened Hamming code, and adding an overall parity-check bit, was proposed for the IBM model 360 mainframe ([Hsia] and Chapter 16 of [LC]).

denote the number of information bits not used, referred to as the *shortening depth*. Let C denote a cyclic (n, k, d) code. A shortened message is obtained by fixing s (arbitrary) message positions to zero. This leaves $k - s$ positions available for the message bits. Without loss of generality, let the highest positions in a message be set to zero. Then $\bar{u}(x) = u_0 + u_1 x + \cdots + u_{k-1-s} x^{k-1-s}$. The output of a *systematic encoder*, when the input is the message polynomial $\bar{u}(x)$, produces the code polynomial $\bar{v}(x) = x^{n-k}\bar{u}(x) + [x^{n-k}\bar{u}(x) \bmod \bar{g}(x)]$, of degree up to $n - 1 - s$. This shows that the resulting shortened code C_s is a linear $(n - s, k - s, d_s)$ code with $d_s \geq d$. In general, the shortened code C_s is no longer cyclic.

Example 24 Let C denote the cyclic Hamming $(7, 4, 3)$ code with generator polynomial $\bar{g}(x) = 1 + x + x^3$. A new code is derived from C by sending 2 leading zeros followed by two information bits and the same three redundant bits computed by an encoder in C. This process gives a set of codewords that forms a *shortened* linear $(5, 2, 3)$ code.

The fundamental property of a shortened code C_s obtained from a cyclic code is that, although the code is generally no longer cyclic, the same encoder and decoder can be used, after the leading zeros are properly taken into account. In computer simulations, it is easy to simply pad the codewords with zeros followed by the codeword in C_s and use the same encoding and decoding algorithms discussed in the book. This method is widely used in hardware implementations of Reed-Solomon decoders. Alternatively, the leading zeros in a message do not need to be included in the codeword. Instead, the decoder circuit is modified to multiply the incoming received polynomial $\bar{r}(x)$ by x^{n-k+s} modulo $\bar{g}(x)$, instead of x^{n-k} modulo $\bar{g}(x)$ in the conventional decoder. More details on the modified encoder and decoder structures for shortened cyclic code can be found in [PW, LC, Wic], among other references.

Another possible solution is to try to construct other classes of cyclic codes with the desired parameters. Interesting classes of cyclic codes not covered in the text are the *non-primitive* BCH codes [PW] and the Euclidean geometry (EG) and projective geometry (PG) codes [LC]. Yet another possibility is to use a *non-binary* cyclic code, such as a Reed-Solomon code discussed in the next chapter, and to express it in terms of bits. This binary image of a RS code will have the additional feature of being able to correct many bursts of errors. See Chapter 4 for more information.

CRC codes

One of the most popular forms of ECC are the *cyclic redundancy check* codes, or CRC codes. These cyclic codes are used to detect errors in blocks of data. CRC codes are cyclic codes of length $n \leq 2^m - 1$. Typically, CRC codes have generator polynomials of the form $(1 + x)\bar{g}(x)$, where $\bar{g}(x)$ is the generator polynomial of a cyclic Hamming code. Common values of m are 12, 16 and 32. The choice of the generator polynomials is dictated by the *undetected error probability*, which depends on the weight distribution of the code. The computation of the undetected error probability of a cyclic code is tantamount to determining its weight distribution. This has remained an elusive task, even after 50 years of coding theory, with some progress reported in [FKKL, Kaz] and references therein. Below is a list of the most popular generator polynomials of CRC codes, or *CRC polynomials*:

Code	m	$\bar{g}(x)$
CRC-12	12	$x^{12}+x^{11}+x^3+x^2+x+1$
CRC-16	16	$x^{16}+x^{15}+x^2+1$
CRC-CCITT	16	$x^{16}+x^{12}+x^5+1$
CRC-32	32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}$ $+x^{10}+x^8+x^7++x^5+x^4+x^2+x+1$

3.2 General decoding of cyclic codes

Let $\bar{r}(x) = \bar{v}(x) + \bar{e}(x)$, where $\bar{e}(x)$ is the *error polynomial* associated with an error vector produced after transmission over a BSC channel. Then the *syndrome polynomial* is defined as

$$\bar{s}(x) \triangleq \bar{r}(x) \bmod \bar{g}(x) = \bar{e}(x) \bmod \bar{g}(x). \tag{3.8}$$

Figure 18 shows the general architecture of a decoder for cyclic codes. The syndrome polynomial $\bar{s}(x)$ is used to determine the error polynomial $\bar{e}(x)$. Since a cyclic code is first of all a linear code, this architecture can be thought of as a "standard array approach" to the decoding of cyclic codes.

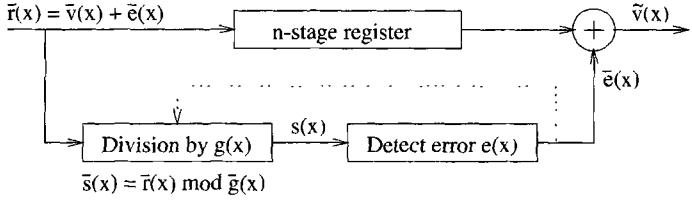


Figure 18 General architecture of a decoder for cyclic codes.

The decoding problem amounts to finding the (unknown) error polynomial $\bar{e}(x)$ from the (known) syndrome polynomial $\bar{s}(x)$. These two polynomials are related by equation (3.8), which is the basis of a *syndrome decoder* (also referred to as a Meggit decoder [Meg]) for cyclic codes. A related decoder is the *error-trapping decoder* [Kasm], which checks if the error polynomial $\bar{e}(x)$ is contained ("trapped") in the syndrome polynomial $\bar{s}(x)$. Only a limited number of classes of codes have relatively simple decoders, e.g., cyclic Hamming and Golay codes. As the error correcting capability $t = \lfloor (d_{min} - 1)/2 \rfloor$ increases, however, the complexity of an architecture based only on (combinatorial) detection of errors becomes too large.

Suppose that an error in the position corresponding to x^{n-1} (the first received bit) occurs. In other words, $\bar{e}(x) = x^{n-1}$. The corresponding syndrome polynomial is $\bar{s}(x) = x^{n-1} \bmod \bar{g}(x)$. The code is cyclic, and thus if an error pattern affecting a given position is detected, any other error can be detected as well, by cyclically shifting the contents of the syndrome polynomial and the error polynomial. The syndrome decoder checks the syndrome for each received position and, if the pattern $x^{n-1} \bmod \bar{g}(x)$ is detected, that position is corrected.

Example 25 In this example, the decoding of a cyclic $(7, 4, 3)$ Hamming code is illustrated. For this code, $\bar{g}(x) = x^3 + x + 1$. The syndrome decoding circuit is shown in Figure 19. The

received bits are stored in a shift register and at the same time fed to a divide-by-$\bar{g}(x)$ circuit. After all the seven bits have been received, the shift register contents are shifted one at a time, and a combinatorial gate checks if the syndrome polynomial $x^6 \bmod (1 + x + x^3) = 1 + x^2$, or (101) in binary vector notation, is present in the shift register when the output of the gate is equal to one, the error is at the position x^6 and is corrected. At the same time, the error is fed back to the divide-by-$\bar{g}(x)$ circuit to bring the contents of the register equal to all zeros, upon successful completion of decoding. This also allows detection of any anomalies at the end of the decoding process, by checking that the contents of the shift register are not all equal to zero.

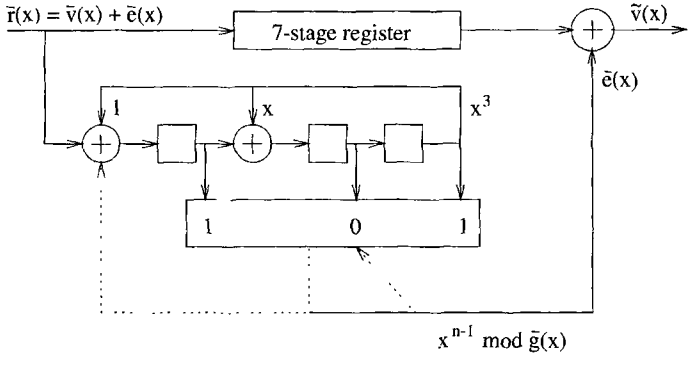


Figure 19 Syndrome decoder for a binary cyclic Hamming (7,4) code.

Attention is now focused on cyclic codes with large error correcting capabilities, for which the decoding problem can be treated as that of solving sets of equations. Because of this, the notion of a *field*, a set in which one can multiply, add and find inverses, is required. Cyclic codes have a rich algebraic structure. It will be shown later that powerful decoding algorithms can be implemented efficiently when the roots of the generator polynomial are invoked and arithmetic over a *finite field* used.

Recall that the generator polynomial is the product of binary irreducible polynomials:

$$\bar{g}(x) = \prod_{j \in J \subset \{1,2,\cdots,\ell\}} \phi_j(x).$$

The algebraic structure of a cyclic codes enables one to find the factors (roots) of each $\phi_j(x)$ in a *splitting field* (also known as extension field). In the case of interest, that is, when the underlying symbols are bits, the splitting field becomes a *Galois field*[3]. Some authors refer to Galois fields as *finite fields*. The standard notation that will be used in the text is $GF(q)$, where $q = 2^m$. (Although, in general, q can be the power of any prime number.)

Example 26 In this example, the reader is reminded that the concept of splitting field is very familiar. Consider the field of real numbers. Over this field, it is well known that the

[3] After the famous French mathematician Evariste Galois (1811-1832).

polynomial $x^2 + 1$ is irreducible. However, over the *complex field*, it splits into $(x + i)(x - i)$, where $i = \sqrt{-1}$. Thus the complex field is the splitting field of the real field!

3.2.1 $GF(2^m)$ arithmetic

It can be shown, with basic abstract algebra [PW, LC] concepts, that, in the field of binary numbers, any polynomial of degree m can be split over $GF(2^m)$. For the purposes of this book, it is sufficient to learn basic computational aspects of finite fields. Serious readers are urged to study a good textbook on abstract algebra[4].

Decoding with $GF(2^m)$ arithmetic allows replacement of complex combinatorial circuits with practical processor architectures that can solve Equation (3.8) as a set of linear equations. In the following, the necessary tools to solve equations involved in decoding of cyclic codes are introduced.

Important properties of $GF(2^m)$

The field $GF(2^m)$ is isomorphic (with respect to "+") to the linear space $\{0,1\}^m$. In other words, for every element $\beta \in GF(2^m)$, there exists a unique m-dimensional binary vector $\bar{v}_\beta \in \{0,1\}^m$.

There is a *primitive element* $\alpha \in GF(2^m)$, such that every nonzero element β in $GF(2^m)$ can be expressed as $\beta = \alpha^j$, $0 \leq j \leq 2^m - 2$. This element α is the root of an irreducible polynomial, called a *primitive polynomial*, $p(x)$ over $\{0,1\}$, i.e., $p(\alpha) = 0$. A primitive element α of the field $GF(2^m)$ satisfies the equation $\alpha^{2^m-1} = 1$, and $n = 2^m - 1$ is the smallest positive integer such that $\alpha^n = 1$.

Example 27 Let $p(x) = x^3 + x + 1$, a primitive polynomial of $GF(2^3)$. Let α be a primitive element such that $p(\alpha) = \alpha^3 + \alpha + 1 = 0$ and $\alpha^7 = 1$. The table below shows three different ways to express, or representations of, elements in $GF(2^3)$.

Power	Polynomial	Vector
—	0	000
1	1	001
α	α	010
α^2	α^2	100
α^3	$1 + \alpha$	011
α^4	$\alpha + \alpha^2$	110
α^5	$1 + \alpha + \alpha^2$	111
α^6	$1 + \alpha^2$	101

When adding elements in $GF(2^m)$, the vector representation is the most useful, because a simple exclusive-or operation is needed. However, when elements are to be multiplied, the power representation is the most efficient. Using the power representation, a multiplication becomes simply an addition modulo $2^m - 1$. The polynomial representation may be appropriate when making operations modulo a polynomial. An example of the need of this polynomial representation was seen in the discussion of shortened cyclic codes, where the value of $x^{n-1} \bmod \bar{g}(x)$ was required.

---

[4] The author likes [Her].

In the power representation, because $\alpha^{2^m-1} = 1$ holds, note that $\alpha^{2^m} = \alpha\alpha^{2^m-1} = \alpha$, $\alpha^{2^m+1} = \alpha^2\alpha^{2^m-1} = \alpha^2$, and so on. This is to say that the powers of α are to be computed modulo $2^m - 1$. Applying the same argument shows that $\alpha^{-1} = \alpha^{-1+2^m-1} = \alpha^{2^m-2}$. For Example 27 above, $\alpha^{-1} = \alpha^{2^3-2} = \alpha^6$. In general, the inverse $\beta^{-1} = \alpha^k$ of an element $\beta = \alpha^\ell$ is found by determining the integer k, $0 \leq k < 2^m - 1$ such that $\alpha^{\ell+k} = 1$, which can be expressed as $\ell + k = 0 \mod (2^m - 1)$. Therefore, $\ell = 2^m - 1 - k$. Also, in the polynomial representation, the equation $p(\alpha) = 0$ is used in order to reduce the expressions. In Example 27, $\alpha^3 = \alpha^3 + 0 = \alpha^3 + (\alpha^3 + \alpha + 1) = \alpha + 1$.

Log and anti-log tables

A convenient way to perform *both* multiplications and additions in $GF(2^m)$ is to use two look-up tables, with different interpretations of the address. This allows one to change between polynomial (vector) representation and power representations of an element of $GF(2^m)$.

The anti-log table $A(i)$ is useful when performing additions. The table gives the value of a binary vector, represented as an integer in natural representation, $A(i)$, that corresponds to the element α^i. The log table $L(i)$ is used when performing multiplications. This table gives the value of a power of alpha, $\alpha^{L(i)}$ that corresponds to the binary vector represented by the integer i. The following equality holds:

$$\alpha^{L(i)} = A(i).$$

The best way to understand how to use the tables in the computation of arithmetic operations in $GF(2^m)$ is through an example.

Example 28 Consider $GF(2^3)$ with $p(\alpha) = \alpha^3 + \alpha + 1$, and $\alpha^7 = 1$. The log and anti-log tables are the following:

Address i	$GF(2^m)$-to-vector Anti-log table, $A(i)$	Vector-to-$GF(2^m)$ Log table, $L(i)$
0	1	-1
1	2	0
2	4	1
3	3	3
4	6	2
5	7	6
6	5	4
7	0	5

Consider the computation of an element $\gamma = \alpha(\alpha^3 + \alpha^5)^3$ in vector form. Using the properties of $GF(2^3)$, γ can be computed as follows: $\alpha^3 + \alpha^5 = 110 \oplus 111 = 001 = \alpha^2$. Thus, $\gamma = \alpha(\alpha^2)^3 = \alpha^{1+6} = \alpha^7 (= 1)$.

On the other hand, using the log and anti-log tables, the computation of γ proceeds as follows: $\gamma = A(L(A(3) \oplus A(5)) * 3 + 1) = A(L(3 \oplus 7) * 3 + 1) = A(L(4) * 3 + 1) = A(2 * 3 + 1) = A(7) = (A(0) = 1)$. In the last step, use was made of the fact that $\alpha^7 = 1$.

Anti-log and log tables are used in performing addition and multiplication over $GF(2^m)$. Computer programs are available on the ECC web site for simulating encoding and decoding algorithms of BCH and Reed-Solomon codes, with arithmetic in $GF(2^m)$. These algorithms are described in subsequent sections.

More properties of $GF(2^m)$

The *minimal polynomial* $\phi_i(x)$ of an element α^i is the smallest degree polynomial that has α^i as a root. The following properties regarding minimal polynomials can be shown. The minimal polynomial $\phi_i(x)$ has binary coefficients and is irreducible over $GF(2) = \{0, 1\}$. Moreover, $\phi_i(x)$ has roots $\alpha^i, \alpha^{2i}, \cdots, \alpha^{2^{\kappa-1}i}$, where κ divides m. These elements are known as the *conjugate elements* of α^i in $GF(2^m)$. The powers of the conjugate elements form a *cyclotomic coset* (see [MS], p. 104, and [GG], p. 391):

$$C_i \triangleq \{i, 2i, 4i, \cdots, 2^{\kappa-1}i\}.$$

Cyclotomic cosets (also called cycle sets in [PW], p. 209) have the property that they partition the set $\mathcal{I}_{2^m-1}$ of integers modulo $2^m - 1$. That is, cyclotomic cosets are disjoint, that is, their intersection is the empty set $C_i \cap C_j = \emptyset$, $i \neq j$, and and the union of all cyclotomic cosets $\bigcup_i C_i = \mathcal{I}_{2^m-1}$.

Example 29 The cyclotomic sets modulo 7 are:

$$\begin{aligned} C_0 &= \{0\} \\ C_1 &= \{1, 2, 4\} \\ C_3 &= \{3, 6, 5\} \end{aligned}$$

The primitive element α of $GF(2^m)$ satisfies the equation $\alpha^{2^m-1} = 1$, and all elements can be expressed as powers of α. From this it follows that the polynomial $(x^{2^m-1} + 1)$ factors over the binary field as

$$(x^{2^m-1} + 1) = \prod_{j=0}^{M} \phi_{\ell_j}(x),$$

and splits completely over $GF(2^m)$ as

$$(x^{2^m-1} + 1) = \prod_{j=0}^{2^m-2} (x + \alpha^j). \tag{3.9}$$

The *order* n_i of an element $\beta = \alpha^i$ of $GF(2^m)$ is the smallest positive integer such that $\beta^{n_i} = 1$. The order n_i of every element in $GF(2^m)$ divides $2^m - 1$.

Importantly, the degree of a minimal polynomial $\phi_i(x)$ is equal to the cardinality (number of elements) of the cyclotomic coset C_i,

$$\deg[\phi_i(x)] = |C_i|.$$

This suggests the following method of finding all factors of $(x^{2^m-1} + 1)$:

1. Generate the cyclotomic cosets modulo $2^m - 1$.
2. For each cyclotomic coset C_s, compute the minimal polynomial $\phi_s(x)$ as the product of linear factors $(x - \alpha^{i_s})$, where $i_s \in C_s$,

$$\phi_s(x) = \prod_{i_s \in C_s} (x + \alpha^{i_s}). \tag{3.10}$$

This method can be used in computing the generator polynomial of any cyclic code of length $n = 2^m - 1$. It is used in the computer simulation programs for BCH codes available on the ECC web site, to compute the generator polynomial given the zeros of the code.

Example 30 Consider $GF(2^3)$ with $p(x) = x^3 + x + 1$. The roots of each of the factors of the polynomial $x^7 + 1$ are shown in the following table. The reader is invited to verify that in fact the products of the linear factors in (3.10) give the resulting binary polynomials.

$\mathcal{C}_s$	Conjugate elements	Minimal polynomial, $\phi_s(x)$
$\mathcal{C}_0 = \{0\}$	1	$\phi_0(x) = x + 1$
$\mathcal{C}_1 = \{1, 2, 4\}$	α,α^2,α^4	$\phi_1(x) = x^3 + x + 1$
$\mathcal{C}_3 = \{3, 6, 5\}$	α^3,α^6,α^5	$\phi_3(x) = x^3 + x^2 + 1$

3.3 Binary BCH codes

BCH codes are cyclic codes that are constructed by specifying their zeros, i.e., the roots of their generator polynomials:

> *A BCH code of $d_{min} \geq 2t_d + 1$ is a cyclic code whose generator polynomial $\bar{g}(x)$ has $2t_d$ consecutive roots α^b, α^b, α^{b+1}, α^{b+2t_d-1}.*

Therefore, a binary BCH (n, k, d_{min}) code has a generator polynomial

$$\bar{g}(x) = LCM\{\phi_b(x), \phi_{b+1}(x), \cdots, \phi_{b+2t_d-1}(x)\},$$

length $n = LCM\{n_b, n_{b+1}, \cdots, n_{b+2t_d-1}\}$, and dimension $k = n - \deg[\bar{g}(x)]$. A binary BCH code has a *designed minimum distance* equal to $2t_d + 1$. However, it should be noted that its true minimum distance may be larger.

Example 31 With $GF(2^3)$, $p(x) = x^3 + x + 1$, $t_d = 1$ and $b = 1$, the polynomial

$$\bar{g}(x) = LCM\{\phi_1(x), \phi_2(x)\} = x^3 + x + 1,$$

generates a binary BCH (7,4,3) code. (This is actually a binary cyclic Hamming code!) Note that the Hamming weight of $\bar{g}(x)$ is equal to 3, so that — in this case, but not always — the designed distance is equal to the true minimum distance of the code.

Example 32 Consider $GF(2^4)$, $p(x) = x^4 + x + 1$, with $t_d = 2$ and $b = 1$. Then

$$\begin{aligned} \bar{g}(x) &= LCM\{\phi_1(x), \phi_3(x)\} \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) \\ &= x^8 + x^7 + x^6 + x^4 + 1 \end{aligned}$$

generates a double-error-correcting binary BCH (15,7,5) code.

Example 33 With $GF(2^4)$, $p(x) = x^4 + x + 1$, $t_d = 3$ and $b = 1$, the polynomial

$$\begin{aligned} \bar{g}(x) &= LCM\{\phi_1(x), \phi_3(x)\phi_5(x)\} \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) \\ &\quad (x^2 + x + 1) \\ &= x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1 \end{aligned}$$

generates a triple-error-correcting binary BCH (15,5,7) code.

The lower bound on the minimum distance of a BCH code, known as the *BCH bound*, is derived next. This is useful not only to estimate the error correcting capabilities of cyclic codes in general, but also serves to point out particular features of BCH codes. Note that the elements $\alpha^b, \alpha^{b+1}, \cdots, \alpha^{b+2t_d-1}$ are roots of the generator polynomial $\bar{g}(x)$, and that every codeword $\bar{v}$ in the BCH code is associated with a polynomial $\bar{v}(x)$ which is a multiple of $\bar{g}(x)$. It follows that

$$\bar{v}(x) \in C \iff \bar{v}(\alpha^i) = 0, \quad b \leq i < b + 2t_d. \tag{3.11}$$

Codeword $\bar{v}$ then satisfies the following set of $2t_d$ equations, expressed in matrix form, based on (3.11):

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha^b & \alpha^{b+1} & \cdots & \alpha^{b+2t_d-1} \\ (\alpha^b)^2 & (\alpha^{b+1})^2 & \cdots & (\alpha^{b+2t_d-1})^2 \\ \vdots & \vdots & \ddots & \vdots \\ (\alpha^b)^{n-1} & (\alpha^{b+1})^{n-1} & \cdots & (\alpha^{b+2t_d-1})^{n-1} \end{pmatrix} \begin{pmatrix} v_0 & v_1 & v_2 & \cdots & v_{n-1} \end{pmatrix} = \bar{0}. \tag{3.12}$$

Consequently, a parity-check matrix of a binary cyclic BCH code is given by

$$H = \begin{pmatrix} 1 & \alpha^b & (\alpha^b)^2 & \cdots & (\alpha^b)^{n-1} \\ 1 & \alpha^{b+1} & (\alpha^{b+1})^2 & \cdots & (\alpha^{b+1})^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{b+2t_d-1} & (\alpha^{b+2t_d-1})^2 & \cdots & (\alpha^{b+2t_d-1})^{n-1} \end{pmatrix} \tag{3.13}$$

This matrix H has the characteristic that every $2t_d \times 2t_d$ submatrix (formed by an arbitrary set of $2t_d$ columns of H) is a *Vandermonde matrix* (see, e.g, [GG], p. 95). Therefore (see Section 2.1), any $2t_d$ columns of H are linearly independent, from which it follows that the minimum distance of the code is $d \geq 2t_d + 1$. ([PW] p. 270, [MS] p. 201, and [LC] p. 149.) Another interpretation of the results above is the following:

(BCH bound.) If the generator polynomial $\bar{g}(x)$ of a cyclic (n, k, d) code has ℓ consecutive roots, say $\alpha^b, \alpha^{b+1}, \cdots, \alpha^{b+\ell-1}$, then $d \geq 2\ell + 1$.

3.4 Polynomial codes

The important class of cyclic polynomial codes includes cyclic RM codes, BCH and Reed-Solomon codes, and finite-geometry codes [KLP, PW, LC]. Polynomial codes are also specified by setting conditions on their zeros:

Let α be a primitive element of $GF(2^{ms})$. Let s be a positive integer, and b a divisor of $2^s - 1$. Then α^h is a root of the generator polynomial $g(x)$ of a μ-th order polynomial code if and only if b divides h and

$$\min_{0 \leq \ell < s} W_{2^s}(h2^\ell) = jb, \text{ with } 0 < j < \left\lceil \frac{m}{b} \right\rceil - \mu,$$

where for an integer i, $i = \sum_{\ell=0}^{m-1} i_\ell 2^{s\ell}$, $W_{2^s}(i)$ is defined as the 2^s-ary weight of integer i,

$$W_{2^s}(i) = \sum_{i=0}^{m-1} i_\ell.$$

According to this definition, both BCH and Reed-Solomon codes are polynomial codes with $b = m = 1$. Reed-Muller codes are subcodes of polynomial codes with $s = 1$. Finite-geometry codes ([LC], Chapter 8) occur as *dual codes of polynomial codes* [PW]. Below, following [LC], the specifications of the zeros of finite-geometry codes are presented.

Euclidean geometry (EG) codes

Let α be a primitive element of $GF(2^{ms})$. Let h be a nonnegative integer less that $2^{ms} - 1$. Then α^h is a root of the generator polynomial $g(x)$ of a (μ, s)-order EG code of length $2^{ms} - 1$ if and only if

$$0 < \max_{0 \le \ell < s} W_{2^s}(h^{(\ell)}) \le (m - \mu - 1)(2^s - 1),$$

where $h^{(\ell)} = 2^\ell h$ modulo $2^{ms} - 1$.

For $s = 1$, EG codes become cyclic $\mathrm{RM}^\star_{m,\mu}$ codes, and therefore EG codes are regarded as generalized RM codes.

Projective geometry (PG) codes

Let α be a primitive element of $GF(2^{(m+1)s})$. Let h be a nonnegative integer less that $2^{(m+1)s} - 1$. Then α^h is a root of the generator polynomial $g(x)$ of a (μ, s)-order PG code of length $n = (2^{ms} - 1)/(2^s - 1)$ if and only if h is divisible by $2^s - 1$ and

$$0 < \max_{0 \le \ell < s} W_{2^s}(h^{(\ell)}) \le j(2^s - 1),$$

where $h^{(\ell)} = 2^\ell h$ modulo $2^{ms} - 1$, and $0 \le j \le m - u$.

3.5 Decoding of binary BCH codes

The main idea in decoding binary BCH codes is to use the elements $\beta \in GF(2^m)$ to *number the positions* of a codeword (or, equivalently, the order of the coefficients of the associated polynomial). This numbering illustrated in Figure 20 for a vector $\bar{r} = (r_0 \quad r_1 \quad \cdots r_{n-1})$ with corresponding $\bar{r}(x)$.

values	r_0	r_1		r_{n-1}
positions	1	α		α^{n-1}

Figure 20 Codeword position numbering using elements of $GF(2^m)$.

Using $GF(2^m)$ arithmetic, the positions of the errors can be found, by solving a set of equations. These equations can be obtained from the error polynomial $\bar{e}(x)$ and the zeros of the code, α^j, for $b \le j \le b + 2t_d - 1$, as shown in the following.

Let $\bar{r}(x) = \bar{v}(x) + \bar{e}(x)$ represent the polynomial associated with a received codeword, where the *error polynomial* is defined as

$$\bar{e}(x) = e_{j_1} x^{j_1} + e_{j_2} x^{j_2} + \cdots + e_{j_\nu} x^{j_\nu}, \tag{3.14}$$

and $\nu \leq t_d$ is the number of errors. The sets $\{e_{j_1}, e_{j_2}, \cdots, e_{j_\nu}\}$, and $\{\alpha^{j_1}, \alpha^{j_1}, \cdots, \alpha^{j_1}\}$ are known as the *error values* and *error positions*, respectively, where $e_j \in \{0, 1\}$ for binary BCH codes[5] and $\alpha \in GF(2^m)$.

The *syndromes* are the evaluation of $\bar{r}(x)$ at each of the zeros of the code:

$$\begin{aligned} S_1 &= r(\alpha^b) = e_{j_1}\alpha^{bj_1} + \cdots + e_{j_\nu}\alpha^{bj_\nu} \\ S_2 &= r(\alpha^{b+1}) = e_{j_1}\alpha^{(b+1)j_1} + \cdots + e_{j_\nu}\alpha^{(b+1)j_\nu} \\ &\vdots \qquad\qquad \vdots \\ S_{2t_d} &= r(\alpha^{b+2t_d-1}) = e_{j_1}\alpha^{(b+2t_d-1)j_1} + \cdots + e_{j_\nu}\alpha^{(b+2t_d-1)j_\nu} \end{aligned}$$

and are equivalent to $\bar{s} = H\bar{r}^\top$, with H given by (3.13).

Let the *error locator polynomial* be defined as

$$\sigma(x) \triangleq \prod_{\ell=1}^{\nu}(1 + \alpha^{j_\ell}x) = 1 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_\nu x^\nu, \tag{3.15}$$

with roots equal to the *inverses of the error locations*. Then the following relation between the coefficients of $\sigma(x)$ and the syndromes holds (see, e.g., [Pet], [LC] p. 154, [PW] p. 284):

$$\begin{pmatrix} S_{\nu+1} \\ S_{\nu+2} \\ \vdots \\ S_{2\nu} \end{pmatrix} = \begin{pmatrix} S_1 & S_2 & \cdots & S_\nu \\ S_2 & S_3 & \cdots & S_{\nu+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-1} \end{pmatrix} \begin{pmatrix} \sigma_\nu \\ \sigma_{\nu-1} \\ \vdots \\ \sigma_1 \end{pmatrix}. \tag{3.16}$$

Solving the *key equation* given by (3.16) constitutes the most computationally intensive operation in decoding BCH codes. Common methods to solve the key equation are:

1. *Berlekamp-Massey algorithm (BMA)*
   The BMA was invented by Berlekamp [Ber1] and Massey [Mas2]. This is a computationally efficient method to solve the key equation, in terms of the number of operations in $GF(2^m)$. The BMA is a popular choice to simulate or implement BCH and RS decoders in software.
2. *Euclidean algorithm (EA)*
   This method to solve the key equation, in polynomial form, was introduced in [SKHN] and further studied in [Man]. Due to its regular structure, the EA is widely used in hardware implementations of BCH and RS decoders.
3. *Direct solution*
   Proposed first by Peterson [Pet], this method directly finds the coefficients of $\sigma(x)$, by solving (3.16) as a set of linear equations. The term *PGZ (Peterson-Gorenstein-Zierler) decoder* is often used in the literature. This is because in [GZ] Peterson's method is applied in decoding nonbinary BCH codes, or RS codes. Naturally, as the complexity of inverting a matrix grows with the cube of the error-correcting capability, the direct solution method works only for small values of t_d. Solutions to (3.16) up to $t_d \leq 6$ are given in [ML], sec. 5.3.

[5] It will be seen later that for Reed-Solomon codes $e_j \in GF(2^m)$.

3.5.1 *General decoding algorithm for BCH codes*

Figure 21 shows the block diagram of a decoder for BCH codes (both binary and non-binary). The decoder consists of digital circuits and processing elements to accomplish the following tasks:

- Compute the syndromes, by evaluating the received polynomial at the zeros of the code [6]

$$S_i \triangleq \bar{r}(\alpha^i), \quad i = b, b+1, \cdots, b+2t_d-1. \tag{3.17}$$

- Find the coefficients of the *error locator* polynomial $\sigma(x)$
- Find the *inverses of the roots of* $\sigma(x)$, i.e., the locations of the errors, $\alpha^{j_1}, \cdots, \alpha^{j_\nu}$.
- Find the *values of the errors* $e_{j_1}, \cdots, e_{j_\nu}$. (Not needed for binary codes.)
- *Correct the received word* with the error locations and values found.

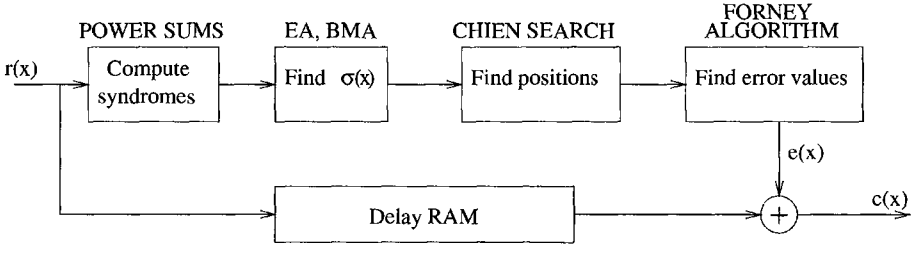


Figure 21 Architecture of a BCH decoder with $GF(2^m)$ arithmetic.

One of the advantages gained in introducing $GF(2^m)$ arithmetic is that the decoding operations can be implemented with relatively simple circuitry and processing elements. As an example, Figure 22 shows how the computation of a syndrome S_j can be implemented in hardware. The multiplier is over $GF(2^m)$, and can be constructed with relatively simple combinatorial logic. Some details on the hardware design of processing elements over $GF(2^m)$ can be found, for example, in [PW], and Chapters 5 and 10 of [WB].

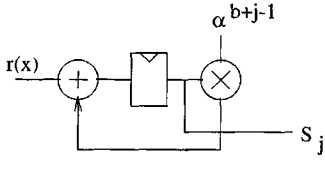


Figure 22 Example of a circuit for computing a syndrome.

[6] For binary BCH codes, it holds that $S_{2i} = S_i^2$, which is useful in reducing the number of computations.

3.5.2 *The Berlekamp-Massey algorithm (BMA)*

The BMA is best understood as an iterative procedure to construct a minimum linear feedback shift-register (LFSR) structure, like the one shown in Figure 23, that reproduces the known *syndrome sequence* $S_1, S_2, \cdots, S_{2t_d}$.

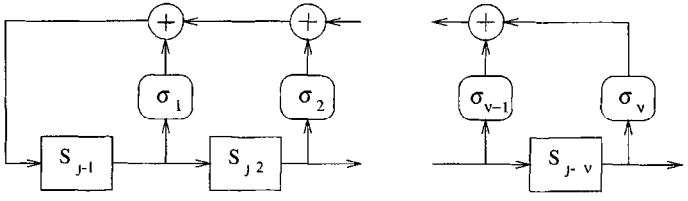


Figure 23 LFSR with taps $\sigma_1, \sigma_2, \ldots \sigma_\nu$ and output $S_1, S_2, \ldots, S_{2\nu}$.

The goal of the BMA is to find a (connection) polynomial $\sigma^{(i+1)}(x)$ of minimal degree that satisfies the following equations, derived from (3.16):

$$\sum_{j=0}^{\ell_{i+1}} S_{k-j}\sigma_j^{(i+1)} = 0, \qquad \ell_i < k < i+1. \tag{3.18}$$

This is equivalent to requiring that $\sigma^{(i+1)}(x) = 1 + \sigma_1^{(i+1)}x + \cdots + \sigma_{\ell_{i+1}}^{(i+1)}x^{\ell_{i+1}}$ be the LFSR connection polynomial that produces the partial sequence of syndromes.

The *discrepancy* at iteration i defined as

$$d_i = S_{i+1} + S_i\sigma_1^{(i)} + \cdots + S_{i-\ell_i+1}\sigma_{\ell_i}^{(i)},$$

measures how well the LFSR reproduces the syndrome sequence, and constitutes a correction term used in computing the value of $\sigma^{(i+1)}$ in the next iteration. There are two cases to consider in the algorithm [Pet][7]:

- If $d_i = 0$, then the equations (3.18) are satisfied for

$$\sigma^{(i+1)}(x) = \sigma^i(x), \qquad \ell_{i+1} = \ell_i. \tag{3.19}$$

- If $d_i \neq 0$: Let $\sigma^{(m)}(x)$ be the solution at iteration m, such that $-1 \leq m < i$, $d_m \neq 0$, and $(m - \ell_m)$ is maximal. Then

$$\begin{aligned} \sigma^{(i+1)}(x) &= \sigma^i(x) + d_i d_m^{-1} x^{i-m}\sigma^{(m)}(x) \\ \ell_{i+1} &= \max\{\ell_i, \ell_m + i - m\}. \end{aligned} \tag{3.20}$$

[7] There is a variation of the BMA, inspired by [Mas2], that is used in some references. This variation will be described in the next chapter. Of course, both versions of the BMA will give the same result!

With an initial value of $i = 0$, the computation of $\sigma^{(i+1)}(x)$ continues until either $i \geq \ell_{i+1} + t_d - 1$ or $i = 2t_d - 1$, or both conditions, are satisfied.

The initial conditions of the algorithm are:

$$\begin{aligned} \sigma^{(-1)}(x) = 1, \quad \ell_{-1} = 0, \quad d_{-1} = 1, \\ \sigma^{(0)}(x) = 1, \quad \ell_0 = 0, \quad d_0 = S_1. \end{aligned} \tag{3.21}$$

Also note that the BMA has control flow instructions (if-else). For this reason, it is not favored in hardware implementations. However, in terms of number of $GF(2^m)$ operations, it is very efficient. This version of the algorithm is implemented in most of the C programs to simulate BCH codes that can be found on the ECC web site.

Example 34 Let C be the triple-error correcting BCH (15,5,7) code of Example 33. As a reference, to check the numerical computations, the power and vector representations of $GF(2^4)$, with primitive polynomial $p(x) = 1 + x + x^4$, are listed below:

Table of elements of $GF(2^4)$, $p(x) = x^4 + x + 1$.

Power	Vector
0	0000
1	0001
α	0010
α^2	0100
α^3	1000
α^4	0011
α^5	0110
α^6	1100
α^7	1011
α^8	0101
α^9	1010
α^{10}	0111
α^{11}	1110
α^{12}	1111
α^{13}	1101
α^{14}	1001

A generator polynomial for C is $\bar{g}(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$. Suppose that the information polynomial is $\bar{u}(x) = x + x^2 + x^4$. Then the corresponding code polynomial is given by $\bar{v}(x) = x + x^2 + x^3 + x^4 + x^8 + x^{11} + x^{12} + x^{14}$.

Let $\bar{r}(x) = 1 + x + x^2 + x^3 + x^4 + x^6 + x^8 + x^{11} + x^{14}$ be the polynomial associated with a vector $\bar{r} = \bar{v} + \bar{e}$ received after transmission of codeword $\bar{v}$ over a BSC channel. (Vector $\bar{e}$ corresponds to the error polynomial $\bar{e}(x) = 1 + x^6 + x^{12}$. Obviously, the decoder does not know this. However, in computing the syndromes below, this knowledge is used to simplify expressions.)

The syndromes are:

$$\begin{aligned} S_1 &= \bar{r}(\alpha) = 1 + \alpha^6 + \alpha^{12} = \alpha \\ S_2 &= S_1^2 = \alpha^2 \end{aligned}$$

$$\begin{aligned} S_3 &= \bar{r}(\alpha^3) = 1 + \alpha^3 + \alpha^6 = \alpha^8 \\ S_4 &= S_2^2 = \alpha^4 \\ S_5 &= \bar{r}(\alpha^5) = 1 + 1 + 1 = 1 \\ S_6 &= S_3^2 = \alpha \end{aligned}$$

Berlekamp-Massey algorithm:

- **Iteration 0:** Initialize. $\sigma^{(-1)}(x) = 1$, $\ell_{-1} = 0$, $d_{-1} = 1$, $\sigma^{(0)}(x) = 1$, $\ell_0 = 0$, and $d_0 = S_1 = \alpha$.
- **Iteration 1:** $i = 0$, $d_0 = \alpha \neq 0$, $m = -1$ maximizes $(-1 + 0) = -1$ for $d_{-1} \neq 0$.

$$\begin{aligned} &\sigma^{(1)}(x) = \sigma^{(0)}(x) + d_0 d_{-1}^{-1} x^{(0-(-1))} \sigma^{(-1)}(x) = 1 + \alpha x, \\ &\ell_1 = \max\{\ell_0, \ell_{-1} + 0 - (-1)\} = 1, \\ &\ell_1 + 3 - 1 \leq 0 \ ? \text{ No :} \\ &d_1 = S_2 + S_1 \sigma_1^{(1)} = \alpha^2 + \alpha(\alpha) = 0. \end{aligned}$$

- **Iteration 2:** $i = 1$, $d_1 = 0$,

$$\begin{aligned} &\sigma^{(2)}(x) = \sigma^{(1)}(x) = 1 + \alpha x, \\ &\ell_2 = \ell_1, \\ &\ell_2 + 3 - 1 \leq 1 \ ? \text{ No :} \\ &d_2 = S_3 + s_2 \sigma_1^{(1)} = \alpha^8 + \alpha^2(\alpha) = \alpha^{13}. \end{aligned}$$

- **Iteration 3:** $i = 2$, $d_2 = \alpha^{13} \neq 0$, $m = 0$ maximizes $(0 - 0) = 0$ for $d_0 \neq 0$.

$$\begin{aligned} &\sigma^{(3)}(x) = \sigma^{(2)}(x) + d_2 d_0^{-1} x^{(2-0)} \sigma^{(0)}(x) = (1 + \alpha x) + \alpha^{13}(\alpha^{-1}) x^2(1) = 1 + \\ &\ell_3 = \max\{\ell_2, \ell_0 + 2 - 0\} = 2, \\ &\ell_3 + 3 - 1 \leq 2 \ ? \text{ No :} \\ &d_3 = S_4 + S_3 \sigma_1^{(3)} + S_2 \sigma_2^{(3)} = \alpha^4 + \alpha^8(\alpha) + \alpha^2(\alpha^{12}) = 0. \end{aligned}$$

- **Iteration 4:** $i = 3$, $d_3 = 0$,

$$\begin{aligned} &\sigma^{(4)}(x) = \sigma^{(3)}(x) = 1 + \alpha x + \alpha^{12} x^2, \\ &\ell_4 = \ell_3, \\ &\ell_4 + 3 - 1 \leq 3 \ ? \text{ No :} \\ &d_4 = S_5 + S_4 \sigma_1^{(4)} + S_3 \sigma_2^{(4)} = 1 + \alpha^4(\alpha) + \alpha^8 \alpha^{12} = 1. \end{aligned}$$

- **Iteration 5:** $i = 4$, $d_4 = 1 \neq 0$, $m = 2$ maximizes $(2 - 1) = 1$ for $d_2 \neq 0$.

$$\begin{aligned} \sigma^{(5)}(x) &= \sigma^{(4)}(x) + d_4 d_2^{-1} x^{(4-2)} \sigma^{(2)}(x) \\ &= (1 + \alpha x + \alpha^{12} x^2) + (1)(\alpha^{13})^{-1} x^2 (1 + \alpha x) \\ &= 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3, \\ \ell_5 &= \max\{\ell_4, \ell_2 + 4 - 2\} = 3, \\ \ell_5 + 3 - 1 &\leq 4 \ ? \text{ No :} \\ d_5 &= S_6 + S_5 \sigma_1^{(5)} + S_4 \sigma_2^{(5)} + S_3 \sigma_3^{(5)} = \alpha + 1 \cdot \alpha + \alpha^4(\alpha^7) + \alpha^8(\alpha^3) = 0. \end{aligned}$$

- **Iteration 6:** $i = 5$, $d_5 = 0$,

$$\begin{aligned} &\sigma^{(6)}(x) = \sigma^{(5)}(x) = 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3, \\ &\ell_6 = \ell_5 = 3, \\ &\ell_6 + 3 - 1 \leq 3 \ ? \text{ Yes : End.} \end{aligned}$$

Therefore $\sigma(x) = 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3$.

That the odd numbered iterations always gave $d_i = 0$ in the previous example is no accident. For *binary* BCH codes this is always the case. The iterations can proceed with only even values of i, with the same results. The only difference is that the stopping criterion needs to be changed to

$$i \geq \ell_{i+2} + t_d - 2.$$

As a result, decoding complexity is reduced. The reader is invited to solve $\sigma(x)$ for the previous example, using only three iterations.

3.5.3 PGZ decoder

This decoding algorithm was first considered by [Pet]. A solution to the key equation (3.16) is to be found using standard techniques for solving a set of linear equations. This solution gives the coefficients of $\sigma(x)$. The decoding problem is that the number of actual errors is unknown. Therefore, a guess has to be made as to the actual number of errors, ν, in the received word. Assume that not all the syndromes, S_i, $1 \leq i \leq 2t$, are equal to zero. (If all syndromes are zero, then the received word is a codeword and decoding finishes!)

The decoder assumes that the maximum number of errors has occurred, $\nu_{\max} = t_d$, and checks if the determinant Δ_i, for $i = \nu_{\max} = t_d$,

$$\Delta_i = \det \begin{pmatrix} S_1 & S_2 & \cdots & S_i \\ S_2 & S_3 & \cdots & S_{i+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_i & S_{i+1} & \cdots & S_{2i-1} \end{pmatrix} \tag{3.22}$$

is equal to zero. If it is, then a smaller number of errors must have occurred. The value of i is decreased by one, and Δ_i tested again, repeating the procedure if necessary, until $i = 1$. Otherwise, if $\Delta_i \neq 0$, the inverse of the syndrome matrix is computed and the values of $\sigma_1, \sigma_2, \cdots, \sigma_\nu$ are found, where $\nu = i$. In the event that $\Delta_i = 0, i = 1, 2, \cdots, t_d$, decoding is unsuccessful and an uncorrectable error pattern has been detected.

Example 35 In this example, the error locator polynomial for the BCH $(15, 5, 7)$ code in Example 34 is found by the PGZ decoding algorithm. First, assume that $i = t_d = 3$ errors occurred. Then the determinant Δ_3 is computed (using cofactors) as follows:

$$\begin{aligned} \Delta_3 &= \det \begin{pmatrix} \alpha & \alpha^2 & \alpha^8 \\ \alpha^2 & \alpha^8 & \alpha^4 \\ \alpha^8 & \alpha^4 & 1 \end{pmatrix} = \alpha(\alpha^8 + \alpha^8) + \alpha^2(\alpha^2 + \alpha^{12}) + \alpha^8(\alpha^6 + \alpha) \\ &= \alpha^{2+7} + \alpha^{8+11} = \alpha^{14}. \end{aligned}$$

Therefore $\Delta_3 \neq 0$ and three errors are assumed to have occurred. Substituting the syndromes found in Example 34 into the key equation (3.16),

$$\begin{pmatrix} \alpha & \alpha^2 & \alpha^8 \\ \alpha^2 & \alpha^8 & \alpha^4 \\ \alpha^8 & \alpha^4 & 1 \end{pmatrix} \begin{pmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha^4 \\ 1 \\ \alpha \end{pmatrix}. \tag{3.23}$$

Note that $\Delta_3^{-1} = \alpha^{-14} = \alpha$. The solution to (3.23) is found to be:

$$\begin{aligned}
\sigma_3 &= \alpha \det \begin{pmatrix} \alpha^4 & \alpha^2 & \alpha^8 \\ 1 & \alpha^8 & \alpha^4 \\ \alpha & \alpha^4 & 1 \end{pmatrix} = \alpha(\alpha^7 + \alpha^{12}) = \alpha^3, \\
\sigma_2 &= \alpha \det \begin{pmatrix} \alpha & \alpha^4 & \alpha^8 \\ \alpha^2 & 1 & \alpha^4 \\ \alpha^8 & \alpha & 1 \end{pmatrix} = \alpha(\alpha^{11} + \alpha) = \alpha^7, \\
\sigma_1 &= \alpha \det \begin{pmatrix} \alpha & \alpha^2 & \alpha^4 \\ \alpha^2 & \alpha^8 & 1 \\ \alpha^8 & \alpha^4 & \alpha \end{pmatrix} = \alpha(\alpha^{15} + \alpha^{15} + \alpha^{15}) = \alpha.
\end{aligned}$$

It follows that $\sigma(x) = 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3$, which is the same result as that obtained by the BMA in example 34.

3.5.4 Euclidean Algorithm (EA)

This algorithm is a well-known recursive procedure to find the greatest common divisor (GCD) between two polynomials (or integers). Its application to BCH decoding is described next.

Let an *error evaluator polynomial* be defined as $\Lambda(x) = \sigma(x)S(x)$, with a *syndrome polynomial*

$$S(x) = 1 + S_1 x + \cdots + S_{2t_d} x^{2t_d}. \tag{3.24}$$

From Equation (3.16), it follows that

$$\Lambda(x) = \sigma(x)S(x) \bmod x^{2t_d+1}. \tag{3.25}$$

The decoding problem can be translated into finding the polynomial $\Lambda(x)$ satisfying (3.25). This can be achieved by applying the extended EA to the polynomials $r_0(x) = x^{2t_d+1}$ and $r_1(x) = S(x)$, such that if at the j-th step

$$r_j(x) = a_j(x)x^{2t_d+1} + b_j(x)S(x),$$

with $\deg[r_j(x)] \leq t_d$, then $\Lambda(x) = r_j(x)$ and $\sigma_i(x) = b_j(x)$. (Note that there is no interest, from the decoding viewpoint, in polynomial $a_i(x)$.)

The extended Euclidean algorithm for computing the GCD of two polynomials is:

Euclidean algorithm: Compute GCD($r_0(x)$,$r_1(x)$)

- Inputs: $r_0(x), r_1(x), \deg[r_0(x)] \geq \deg[r_1(x)]$
- Initial conditions: $a_0(x) = 1, b_0(x) = 0, a_1(x) = 0, b_1(x) = 1$.
- At step j ($j \geq 2$), apply *long division* to $r_{j-2}(x)$ and $r_{j-1}(x)$,

$$r_{j-2}(x) = q_j(x)r_{j-1}(x) + r_j(x), \qquad 0 \leq \deg[r_j(x)] < \deg[r_{j-1}(x)]$$

- and compute

$$a_j(x) = a_{j-2}(x) - q_j(x)a_{j-1}(x), \quad b_j(x) = b_{j-2}(x) - q_j(x)b_{j-1}(x).$$

- Stop at iteration j_{last} when $\deg\,[r_{j_{last}}(x)] = 0$.

Then $GCD(r_0(x), r_1(x)) = r_k(x)$, with k the largest nonzero integer such that $r_k(x) \neq 0$ and $k < j_{last}$.

Example 36 In this example, the error locator polynomial $\sigma(x)$ for the BCH $(15, 5, 7)$ code in Example 34 is computed using the EA.

- **Initial conditions**

$$\begin{aligned} r_0(x) &= x^7, \\ r_1(x) &= S(x) = 1 + \alpha x + \alpha^2 x^2 + \alpha^8 x^3 + \alpha^4 x^4 + x^5 + \alpha x^6, \\ b_0(x) &= 0, \\ b_1(x) &= 1. \end{aligned}$$

- $j = 2$:

$$x^7 = (1+\alpha x+\alpha^2 x^2+\alpha^8 x^3+\alpha^4 x^4+x^5+\alpha x^6)(\alpha^{14}x+\alpha^{13})+\alpha^8 x^5+\alpha^{12}x^4+\alpha^{11}x^3+\alpha^{13}.$$

$$\begin{aligned} r_2(x) &= \alpha^8 x^5 + \alpha^{12} x^4 + \alpha^{11} x^3 + \alpha^{13}, \\ q_2(x) &= \alpha^{14} x + \alpha^{13}, \text{ and} \\ b_2(x) &= b_0(x) + q_2(x) b_1(x) = \alpha^{14} x + \alpha^{13}. \end{aligned}$$

- $j = 3$:

$$S(x) = (\alpha^8 x^5 + \alpha^{12} x^4 + \alpha^{11} x^3 + \alpha^{13})(\alpha^8 x + \alpha^2) + \alpha^{14} x^4 + \alpha^3 x^3 + \alpha^2 x^2 + \alpha^{11} x.$$

$$\begin{aligned} r_3(x) &= \alpha^{14} x^4 + \alpha^3 x^3 + \alpha^2 x^2 + \alpha^{11} x, \\ q_3(x) &= \alpha^8 x + \alpha^2, \text{ and} \\ b_3(x) &= b_1(x) + q_3(x) b_2(x) = \alpha^7 x^2 + \alpha^{11} x. \end{aligned}$$

- $j = 4$:

$$\alpha^8 x^5 + \alpha^{12} x^4 + \alpha^{11} x^3 + \alpha^{13} = (\alpha^{14} x^4 + \alpha^3 x^3 + \alpha^2 x^2 + \alpha^{11} x)(\alpha^9 x) + \alpha^5 x + \alpha^{13}.$$

$$\begin{aligned} r_4(x) &= \alpha^5 x + \alpha^{13}, \\ q_4(x) &= \alpha^9 x, \text{ and} \\ b_4(x) &= b_2(x) + q_4(x) b_3(x) = \alpha x^3 + \alpha^5 x^2 + \alpha^{14} x + \alpha^{13}. \end{aligned}$$

Because $\deg[r_4(x)] = 1 \leq 3$, the algorithm stops.

It follows that $\sigma(x) = b_4(x) = \alpha^{13}(1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3)$, which has the same roots as the polynomial obtained by the BMA and PGZ decoder (see example 37 below), and differs only by a constant factor[8].

As the example above illustrates, generally the error-locator polynomial obtained with the EA will differ from that obtained by BMA or PGZ decoding by a constant factor. In decoding, however, interest is in the roots of these polynomials and not their coefficients, and thus any of the three methods discussed so far can be used to determine $\sigma(x)$.

---

[8] The normalized polynomial $\sigma_{\text{norm}}(x) = \sigma_0^{-1}\sigma(x)$ is identical to the one obtained in the BMA and PGZ procedures.

3.5.5 *Chien search and error correction*

To find the roots of $\sigma(x)$, a simple trial-and-error procedure — called *Chien search* — is performed. All nonzero elements β of $GF(2^m)$ are generated in sequence 1, α, α^2, ... and the condition $\sigma(\beta^{-1}) = 0$ tested. This process is easy to implement in hardware. Moreover, finding roots (factoring) of polynomials over $GF(2^m)$ is a challenging mathematical problem that remains to be solved.

For binary BCH codes, once the error locations $j_1, ..., j_\nu$ are known, the corresponding bits in the received word are complemented

$$\hat{v}_{j_\ell} = v_{j_\ell} + 1, \qquad 1 \leq \ell \leq \nu$$

and the estimated codeword $\hat{v}(x)$ generated.

Example 37 Continuing with Example 34, the roots of $\sigma(x)$ are 1, $\alpha^9 = \alpha^{-6}$ and $\alpha^3 = \alpha^{-12}$. Stated in a different way, $\sigma(x)$ *factors* as

$$\sigma(x) = (1 + x)(1 + \alpha^6 x)(1 + \alpha^{12} x).$$

Consequently, the estimated error polynomial is $\bar{e}(x) = 1 + x^6 + x^{12}$, and

$$\bar{r}(x) = x + x^2 + x^3 + x^4 + x^6 + x^8 + x^{11} + x^{14}.$$

Three errors have been corrected.

3.5.6 *Errors-and-erasures decoding*

There are many situations in which a decision on a received symbol is not considered reliable. An example is binary transmission over an AWGN channel, with bits mapped to real amplitudes, e.g., BPSK with $0 \mapsto +1$ and $1 \mapsto -1$. If the received values are too close to zero, then it may be more advantageous, from the viewpoint of minimizing the probability of a decoding error, to declare a "no-decision". In such a case, the received symbol is "erased" and it is called an *erasure*[9]. Declaring erasures is the simplest form of *soft-decision*, which will be the focus of attention in Chapter 7.

Introduction of erasures has the advantage, with respect to errors-only decoding, that the *positions are known to the decoder*. Let d be the minimum distance of a code, ν be the number of errors and μ be the number of erasures contained in a received word. Then, the minimum Hamming distance between codewords is reduced to at least $d - \mu$ in the non-erased positions. It follows that the error-correcting capability is $\lfloor (d - \mu - 1)/2 \rfloor$ and the following relation holds

$$d > 2\nu + \mu. \tag{3.26}$$

The above inequality is intuitively satisfying: For a fixed minimum distance, it is twice as difficult to correct an error than to correct an erasure, because the erased positions are already known.

For binary linear codes, including binary BCH codes, erasures can be corrected with the following method:

---

[9] In information theoretical terms, the BSC channel becomes a two-input three-output *binary erasure channel* (BEC) [CoT].

1. Place zeros in all erased positions and decode to a codeword $\hat{v}_0(x)$.
2. Place ones in all erased positions and decode to a codeword $\hat{v}_1(x)$.
3. Chose as decoded word the closest $\hat{v}_j(x)$ to the received word $\bar{r}(x)$ in the non-erased positions, $j = 0, 1$. (Alternatively, the codeword that required the smallest number of error corrections [ML].)

As a result, erasures can be corrected with two rounds of errors-only decoding.

Example 38 Consider a cyclic Hamming $(7, 4, 3)$ code with $\bar{g}(x) = 1 + x + x^3$, and $GF(2^3)$ with a primitive element α such that $p(\alpha) = 1 + \alpha + \alpha^3 = 0$. Suppose that $\bar{v}(x) = x + x^2 + x^4$ is transmitted and that $\mu = 2$ erasures are introduced at the receiver. Since $d = 3 > \mu$, these erasures can be corrected. Let $\bar{r}(x) = f + x + x^2 + fx^3 + x^4$ be polynomial associated with the received vector, where f denotes an erasure.

First decoding ($f = 0$): $\bar{r}_0(x) = x + x^2 + x^4$. The syndromes are $S_1 = \bar{r}_0(\alpha) = 0$, and $S_2 = S_1^2 = 0$. Therefore, $\hat{v}_0(x) = \bar{r}_0(x) = x + x^2 + x^4$.

Second decoding ($f = 1$): $\bar{r}_1(x) = 1 + x + x^2 + x^3 + x^4$, $S_1 = \alpha$ and $S_2 = \alpha^2$. Equation (3.16) in this case is: $\alpha\sigma_1 = \alpha^2$. Therefore $\sigma(x) = 1 + \alpha x$, $\bar{e}(x) = x$ and $\hat{v}_1(x) = \bar{r}_1(x) + \bar{e}(x) = 1 + x^2 + x^3 + x^4$, which differs from $\bar{r}(x)$ in one of the non-erased positions. As a result, $\hat{v}_0(x)$ is selected as the decoded word. Two erasures have been corrected.

3.6 Weight distribution and performance bounds

In general, the weight distribution of a binary linear (n, k) code C can be obtained by enumerating all 2^k code vectors $\bar{v}$ and computing their Hamming weight. Obviously, for large values of k, this is a formidable task. Let A_w denote the number of codewords of Hamming weight w. The *MacWilliams identity* relates the *weight distribution sequence* (WDS) of a linear code, $A(x) \triangleq A_0 + A_1 x + x_2 x^2 + \cdots + A_n x^n$, with the WDS of its *dual*[10] $(n, n-k)$ *code* $C^\perp$, $B(x) \triangleq B_0 + B_1 x + x_2 x^2 + \cdots + B_n x^n$, by

$$A(x) = 2^{-n+k}(1+x)^n B\left[\frac{1-x}{1+x}\right], \tag{3.27}$$

which can also be expressed as

$$B(x) = 2^{-k}(1+x)^n A\left[\frac{1-x}{1+x}\right]. \tag{3.28}$$

Therefore, for high-rate codes, it is simpler to compute the WDS $B(x)$ of the dual code and then use (3.27) to compute $A(x)$. Alternatively, the WDS of a low-rate code is easy to compute and the WDS of the dual code can be obtained using (3.28).

For some classes of codes, the trellis structure can be used[11]. In [DFK2], a trellis-based method for computing the weight distribution of *extended BCH codes* of length 128 is given.

---

[10] Recall that the dual code $C^\perp$ has generator matrix H, where H is the parity-check matrix of C.

[11] More on trellises can be found in Chapter 7.

For shorter lengths, the weight distributions are not difficult to compute using MacWilliams identity. The definition of *extended cyclic code* is introduced next. A binary extended cyclic code is obtained from a binary cyclic code by appending, at the start of each code vector, an *overall parity-check bit*. The extended code of a cyclic code is no longer cyclic. Let H denotes the parity-check of the cyclic code, then the parity-check matrix of the extended code, denoted H_{ext} is given by

$$H_{\text{ext}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & & & \\ 0 & & H & \\ 0 & & & \end{pmatrix}. \tag{3.29}$$

Appendix A lists the weight distributions of all extended binary BCH codes of length up to 128. These data files are available on the ECC web site. The appendix lists only those terms of Hamming weight up to $(n+1)/2$, for $n = 2^m - 1$. In the case of extended BCH codes, it holds that $A^{(\text{ext})}_{n+1-w} = A^{(\text{ext})}_w$. The data is useful in finding the weight distribution of the binary cyclic BCH codes of length up to 127. This can be done by application of the following result (which is obtained as a special case of Theorems 8.14 and 8.15 in [PW]):

Let C be a binary cyclic BCH (n, k) code with WDS $A(x)$, obtained by eliminating the overall parity-check bit in a binary extended BCH $(n+1, k)$ code C_{ext}, with WDS $A^{(\text{ext})}(x)$. Then, for w even,

$$\begin{aligned} (n+1)A_{w-1} &= wA^{(\text{ext})}_w, \\ wA_w &= (n+1-w)A_{w-1} \end{aligned} \tag{3.30}$$

Example 39 Consider the binary extended Hamming $(8, 4, 4)$ code. This code has parity-check matrix (see also Example 23),

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

It can be easily verified that $A^{(\text{ext})}(x) = 1 + 14x^4 + x^8$. To compute the WDS of the binary cyclic Hamming $(7, 4, 3)$, use (3.30) to obtain

$$\begin{aligned} 8A_3 = 4A^{(\text{ext})}_4 \quad &\rightarrow \quad A_3 = 7, \\ 4A_4 = (8-4)A_3 \quad &\rightarrow \quad A_4 = 7, \\ 8A_7 = 8A^{(\text{ext})}_8 \quad &\rightarrow \quad A_7 = 1. \end{aligned}$$

3.6.1 Error performance evaluation

With knowledge of the WDS of a code C, the error performance can be estimated, as discussed in Chapter 1. The WDS and the union bound (1.34) give good estimates of the error performance of code C with binary transmission over an AWGN channel.

As an example, the union bound was evaluated using Appendix A for extended BCH codes of length 8 to 64. The results are shown in Figures 24 to 27. Using the WDS from the appendix

and the union bound for flat Rayleigh fading channels (1.40), with Monte Carlo integration and the term A_w replaced by wA_w/n to account for bit errors, Figure 28 shows union bounds on the bit error rate for extended BCH codes of length 8. Bound for other codes can be computed in the same way.

The main point of this section is that, before choosing a particular soft-decision decoding algorithm (see Chapter 7), the weight distribution and the union bound can give an estimate of the performance that is achievable over a certain channel. For the AWGN channel and the flat Rayleigh fading channel, the union bounds are tight at bit error rates below 10^{-4}.

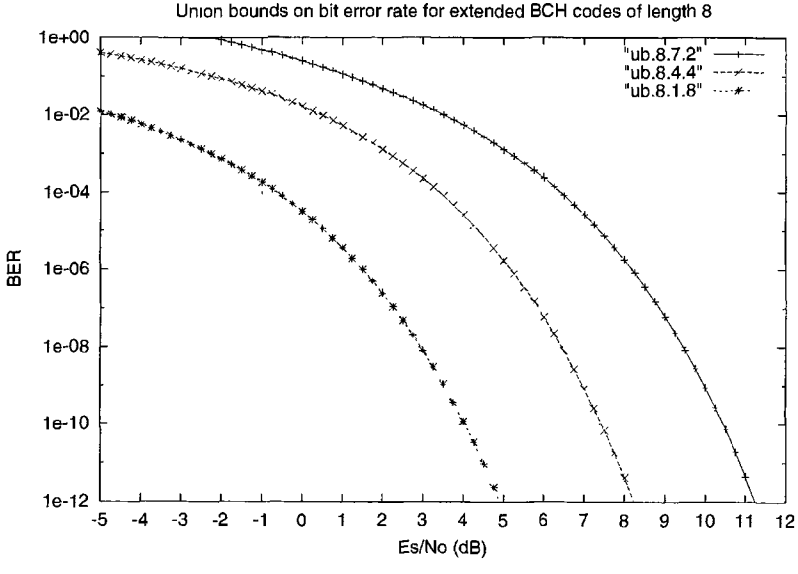


Figure 24 Union bounds on the BER for extended BCH code of length 8. Binary transmission over an AWGN channel.

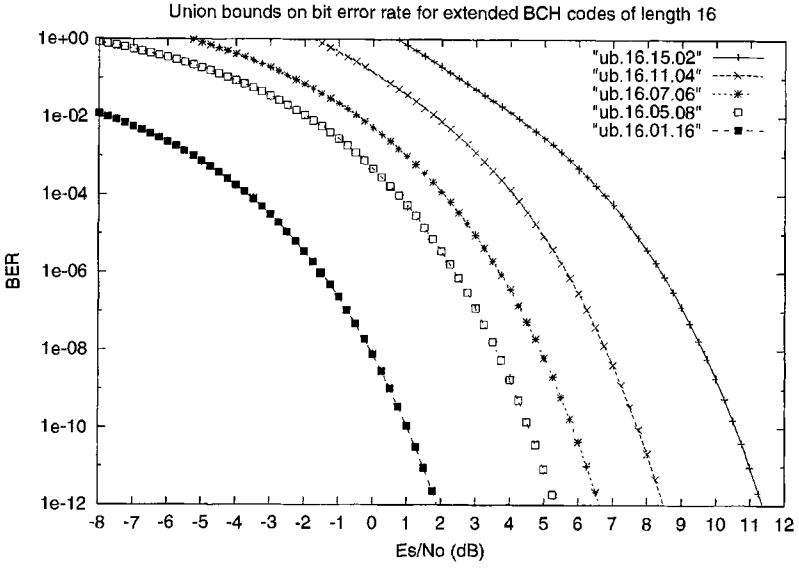


Figure 25 Union bounds on the BER for extended BCH code of length 16. Binary transmission over an AWGN channel.

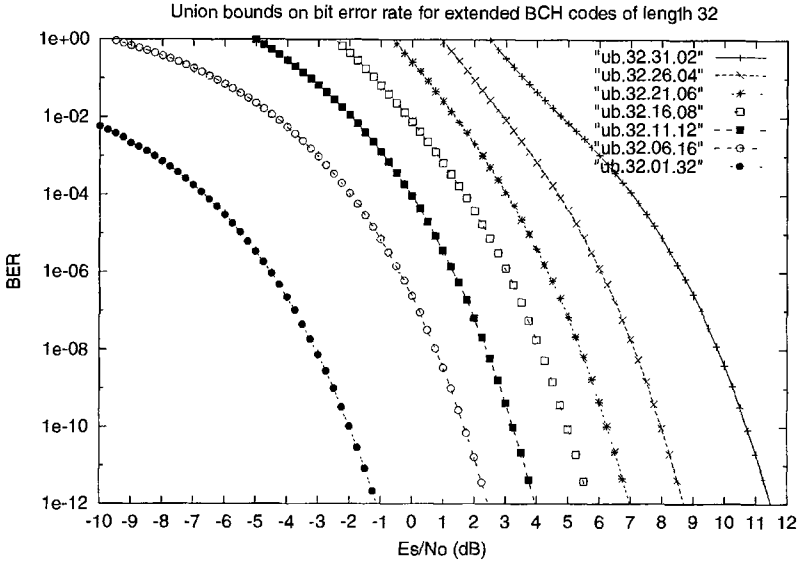


Figure 26 Union bounds on the BER for extended BCH code of length 32. Binary transmission over an AWGN channel.

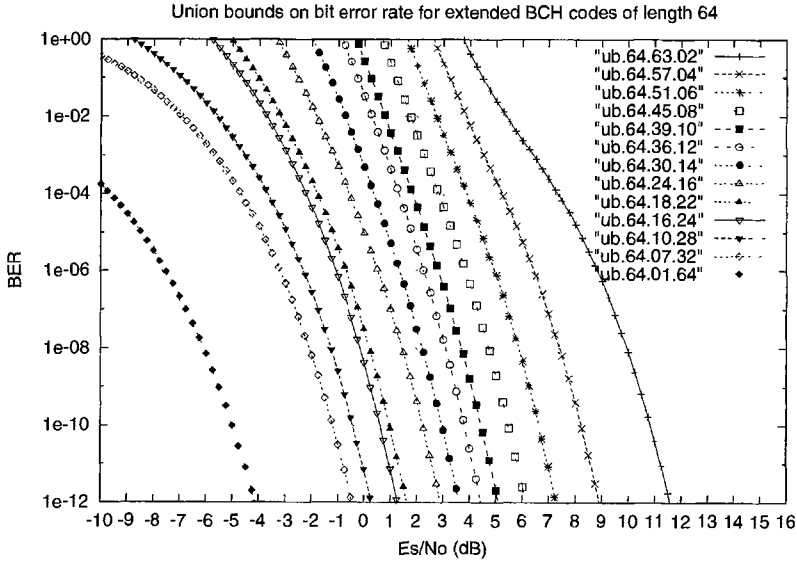


Figure 27 Union bounds on the BER for extended BCH code of length 64. Binary transmission over an AWGN channel.

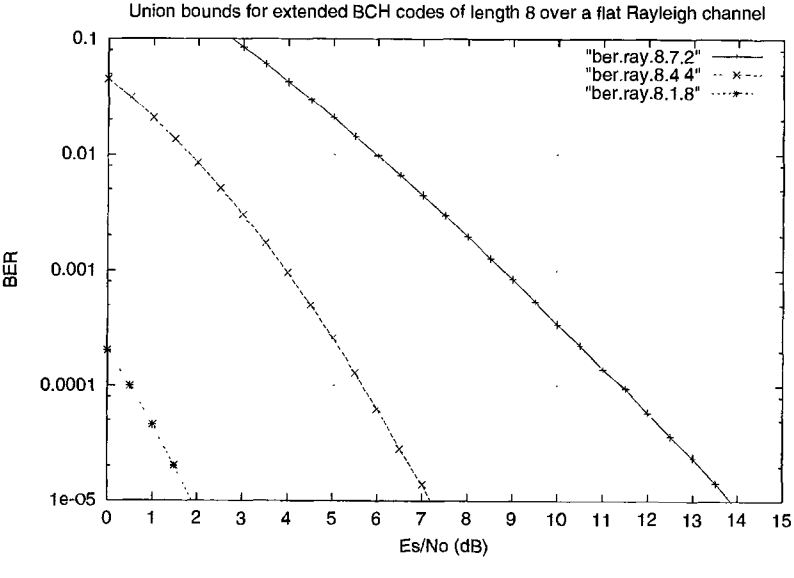


Figure 28 Union bounds on the BER for extended BCH code of length 8. Binary transmission over a flat Rayleigh fading channel.

4

Non-binary BCH codes: Reed-Solomon codes

In this chapter the most celebrated class of ECC schemes is introduced and their encoding and decoding algorithms explained. Reed-Solomon (RS) codes have found numerous applications in digital storage and communication systems. Examples include the famous RS $(255, 223, 33)$ code for NASA space communications, shortened RS codes over $GF(2^8)$ for CD-ROM, DVD and Terrestrial Digital HDTV transmission applications, an extended RS $(128, 122, 7)$ code over $GF(2^7)$ for cable modems, among many others.

4.1 RS codes as polynomial codes

Similar to Reed-Muller codes, RS codes can be defined as codewords with components equal to the evaluation of a certain polynomial. As a matter of fact, this was the way that RS codes were originally defined by Reed and Solomon in [RS]. RM codes, finite-geometry codes [LC] and RS codes are all members of a large class of codes: *Polynomial codes* [PW], which are closely related to *algebraic-geometry (AG) codes* [Pre]. Let

$$\bar{u}(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1} \tag{4.1}$$

be an information polynomial, with $u_i \in GF(2^m)$, $1 \leq i < k$. Clearly, there are 2^{mk} such polynomials. By evaluating (4.1) over the nonzero elements of $GF(2^m)$, a codeword in an RS $(2^m - 1, k, d)$ code of length 2^m is obtained,

$$\bar{v} = (\, u(1) \quad u(\alpha) \quad u(\alpha^2) \quad \cdots \quad u(\alpha^{2^m-2}) \,). \tag{4.2}$$

4.2 From binary BCH to RS codes

RS codes can also be interpreted as *nonbinary* BCH codes. That is, RS codes are BCH codes in which the values of the code coefficient are taken from $GF(2^m)$. In particular, for a t_d-error correcting RS code, the zeros of the code are $2t_d$ consecutive powers of α. Moreover, because over $GF(2^m)$ minimal polynomials are of the form $\phi_i(x) = (x - \alpha^i)$, $0 \leq i < 2^m - 1$, see Equation (3.9), the factors of the generator polynomial are now *linear*, and

$$\bar{g}(x) = \prod_{j=b}^{b+2t_d-1} (x + \alpha^j), \tag{4.3}$$

where b is an integer, usually $b = 0$ or $b = 1$.

It follows from (4.3), and the BCH bound on page 45, that the minimum distance of a Reed-Solomon (n, k, d) code C over $GF(2^m)$ is $d \geq n - k + 1$. On the other hand, the Singleton bound [Sin] $d \leq n - k + 1$ implies that $d = n - k + 1$. A code that satisfies this equality is known as a *maximum distance separable* (MDS) code [Sin]. Therefore RS codes are MDS codes. This gives RS codes useful properties. Among them, shortened RS codes are also MDS codes.

Using the isomorphism between $GF(2^m)$ and $\{0,1\}^m$, for every m-bit vector $\bar{x}_\beta$ there is a corresponding element $\beta \in GF(2^m)$,

$$m\text{-bits} \iff \beta_j \in GF(2^m), \quad 0 \leq j < 2^m - 1.$$

In other words, m information bits can be grouped to form symbols in $GF(2^m)$. Conversely, if the elements of $GF(2^m)$ are expressed as vectors of m bits, then a binary linear code of length and dimension $n = m(2^m - 1)$ and $k = m(2^m - 1 - 2t_d)$, respectively, is obtained. The minimum distance of this code is at least $2t_d + 1$. This *binary image* code can correct, in addition to up to t_d random errors, many *random bursts of errors*. For example, any single burst of up to $m(t_d - 1) + 1$ bits can be corrected. This follows from the fact that a burst of errors of length up to $m(q - 1) + 1$ bits is contained in at most q symbols of $GF(2^m)$. Therefore, there are many combinations of random errors and bursts of errors that an RS code can correct. To a great extent, this is the reason why RS codes are so popular in practical systems.

Example 40 Let $m = 3$, and $GF(2^3)$ be generated by a primitive element α with $p(\alpha) = \alpha^3 + \alpha + 1 = 0$. Let $b = 0$ and $t_d = 2$. Then there is an RS(7,3,5) code C with generator polynomial

$$\begin{aligned} \bar{g}(x) &= (x+1)(x+\alpha)(x+\alpha^2)(x+\alpha^3) \\ &= x^4 + \alpha^2 x^3 + \alpha^5 x^2 + \alpha^5 x + \alpha^6. \end{aligned}$$

By mapping the symbols of $GF(2^3)$ into binary vectors of length 3, code C becomes a binary $(21, 9, 5)$ code that is capable of correcting up to 2 random errors as well as any single burst of up to 4 bits.

4.3 Decoding RS codes

The core of the decoding algorithms of RS codes is similar to that of binary BCH codes. The only difference is that the *error values*, e_{j_ℓ}, $1 \leq \ell \leq \nu$, for $\nu \leq t_d$, have to be computed. In general, this is done using the *Forney algorithm* [For2]. The expression below holds for RS codes with an arbitrary set of $2t_d$ consecutive zeros $\{\alpha^b, \alpha^{b+1}, \cdots \alpha^{b+2t_d-1}\}$,

$$e_{j_\ell} = \frac{(\alpha^{j_\ell})^{2-b} \Lambda(\alpha^{-j_\ell})}{\sigma'(\alpha^{-j_\ell})}, \tag{4.4}$$

where $\sigma'(x)$ represents the formal derivative of $\sigma(x)$ with respect to x. (A similar expression can be found in [RC], p. 276.) The polynomial $\Lambda(x)$ in (4.4) is an *error evaluator polynomial*, which is defined as

$$\Lambda(x) = \sigma(x) S(x) \bmod x^{2t_d+1}. \tag{4.5}$$

Before introducing the first example of RS decoding, an alternative version of the BMA is presented, referred to as the *Massey algorithm* (or MA). The algorithm was invented by Massey in [Mas2], and is also described in [ML, Wic].

Massey algorithm to synthesize an LFSR

1. Initialize the algorithm with $\sigma(x) = 1$ (the LFSR connection polynomial), $\rho(x) = x$ (the correction term), $i = 1$ (syndrome sequence counter), $\ell = 0$ (register length).

2. Get a new syndrome and compute discrepancy:

$$d = S_i + \sum_{j=1}^{\ell} \sigma_j S_{i-j}$$

3. Test discrepancy: $d = 0$? Yes: Go to 8.

4. Modify connection polynomial:

$$\sigma_{\text{new}}(x) = \sigma(x) - d\rho(x)$$

5. Test register length: $2\ell \geq i$? Yes: Go to 7.

6. Change register length and update correction term: Let $\ell = i - \ell$ and $\rho(x) = \sigma(x)/d$

7. Update connection polynomial: $\sigma(x) = \sigma_{\text{new}}(x)$.

8. Update correction term: $\rho(x) = x\rho(x)$.

9. Update syndrome sequence counter: $i = i + 1$.

10. Stopping condition: If $i < d$ go to 2. Else, stop.

Example 41 Let C be the same RS$(7, 3, 5)$ code as in Example 40. Suppose that

$$\bar{r}(x) = \alpha x^2 + \alpha^5 x^4$$

is the received polynomial. Then $S_1 = r(1) = \alpha + \alpha^5 = \alpha^6$, $S_2 = \bar{r}(\alpha) = \alpha^3 + \alpha^2 = \alpha^5$, $S_3 = \bar{r}(\alpha^2) = \alpha^5 + \alpha^6 = \alpha$ and $S_4 = \bar{r}(\alpha^3) = 1 + \alpha^3 = \alpha$. Equation (3.16) gives:

$$\begin{pmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha \end{pmatrix} \begin{pmatrix} \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \alpha \end{pmatrix}.$$

Three methods of finding $\sigma(x)$ are shown below.

Direct solution (PGZ algorithm)

Assume two errors. Then $\Delta_2 = \alpha^7 + \alpha^{10} = 1 + \alpha^3 = \alpha \neq 0$. Therefore two errors must have occurred and

$$\sigma_2 = \alpha^6 \det \begin{pmatrix} \alpha & \alpha^5 \\ \alpha & \alpha \end{pmatrix} = \alpha^6,$$

$$\sigma_1 = \alpha^6 \det \begin{pmatrix} \alpha^6 & \alpha \\ \alpha^5 & \alpha \end{pmatrix} = \alpha,$$

from which it follows that

$$\sigma(x) = 1 + \alpha x + \alpha^6 x^2 = (1 + \alpha^2 x)(1 + \alpha^4 x).$$

Massey algorithm

$$S_1 = \alpha^6,\ S_2 = \alpha^5,\ S_3 = \alpha,\ S_4 = \alpha.$$

- $i = 0$: $\sigma(x) = 1$, $\ell = 0$, $\rho(x) = x$.
- $i = 1$: $d = S_1 = \alpha^6$,

$$\begin{aligned} &\sigma_{\text{new}}(x) = \sigma(x) + d\rho(x) = 1 + \alpha^6 x, \\ &2\ell = 0 < i, \ell = i - \ell = 1, \\ &\rho(x) = \sigma(x)/d = \alpha^{-6} = \alpha, \\ &\rho(x) = x\rho(x) = \alpha x, \qquad \sigma(x) = \sigma_{\text{new}}(x). \end{aligned}$$

- $i = 2$: $d = S_2 + \sum_{j=1}^{1} \sigma_j S_{2-j} = \alpha^5 + \alpha^6\alpha^6 = 0$.

$$\rho(x) = x\rho(x) = \alpha x^2.$$

- $i = 3$: $d = S_3 + \sum_{j=1}^{1} \sigma_j S_{3-j} = \alpha + \alpha^6\alpha^5 = \alpha^2$.

$$\begin{aligned} &\sigma_{\text{new}}(x) = \sigma(x) + d\rho(x) = 1 + \alpha^6 x + \alpha^3 x^2, \\ &2\ell = 2 < i, \ell = i - \ell = 2, \\ &\rho(x) = \sigma(x)/d = \alpha^5 + \alpha^4 x, \\ &\rho(x) = x\rho(x) = \alpha^5 x + \alpha^4 x^2, \qquad \sigma(x) = \sigma_{\text{new}}(x). \end{aligned}$$

- $i = 4$: $d = S_4 + \sum_{j=1}^{2} \sigma_j S_{4-j} = \alpha + \alpha^6\alpha + \alpha^3\alpha^5 = 1$.

$$\begin{aligned} \sigma_{\text{new}}(x) &= \sigma(x) + d\rho(x) = 1 + \alpha^6 x + \alpha^3 x^2 + (1)(\alpha^5 x + \alpha^4 x^2) \\ &= 1 + \alpha x + \alpha^6 x^2, \\ 2\ell &\geq 4 \\ \rho(x) &= x\rho(x) = \alpha^5 x^2 + \alpha^4 x^3, \qquad \sigma(x) = \sigma_{\text{new}}(x). \end{aligned}$$

- $i = 5 > d$. Stop.

Euclidean algorithm

- Initial conditions:

$$\begin{aligned} r_0(x) &= x^5, \\ r_1(x) &= S(x) = 1 + \alpha^6 x + \alpha^5 x^2 + \alpha x^3 + \alpha x^4, \\ b_0(x) &= 0, \\ b_1(x) &= 1. \end{aligned}$$

- $j = 2$:

$$x^5 = (1 + \alpha^6 x + \alpha^5 x^2 + \alpha x^3 + \alpha x^4)(\alpha^6 x + \alpha^6) + \alpha^5 x^3 + x^2 + \alpha x + \alpha^6.$$

$$\begin{aligned} r_2(x) &= \alpha^5 x^3 + x^2 + \alpha x + \alpha^6, \\ q_2(x) &= \alpha^6 x + \alpha^6, \\ b_2(x) &= 0 + (\alpha^6 x + \alpha^6)(1) = \sigma^6 x + \sigma^6. \end{aligned}$$

- $j = 3$:

$$1 + \alpha^6 x + \alpha^5 x^2 + \alpha x^3 + \alpha x^4 = (\alpha^5 x^3 + x^2 + \alpha x + \alpha^6)(\alpha^3 x + \alpha^2) + \alpha^6 x^2 + \alpha x + \alpha^3.$$

$$\begin{aligned} r_3(x) &= \alpha^6 x^2 + \alpha x + \alpha^3, \\ q_3(x) &= \alpha^3 x + \alpha^2, \\ b_3(x) &= 1 + (\alpha^3 x + \alpha^2)(\alpha^6 x + \alpha^6) = \alpha^3 + \alpha^4 x + \alpha^2 x^2. \end{aligned}$$

Algorithm stops, as $\deg[r_3(x)] = 2 = t_d$.

It follows that $\sigma(x) = \alpha^3 + \alpha^4 x + \alpha^2 x^2 = \alpha^3(1 + \alpha x + \alpha^6 x^2)$.

In all the above algorithms, the following error locator polynomial is found, up to a constant term:

$$\sigma(x) = 1 + \alpha x + \alpha^6 x^2 = (1 + \alpha^2 x)(1 + \alpha^4 x).$$

Therefore, the error positions are $j_1 = 2$ and $j_2 = 4$. In a computer program or a hardware implementation, Chien search yields these two values as the (inverse) roots of $\sigma(x)$. Also note that $\sigma'(x) = \alpha$. (Because, in $GF(2^m)$, $2a = a + a = 0$.)

To compute the error values, using either the Berlekamp-Massey algorithm (BMA or MA versions) or the PGZ algorithm, the error evaluator polynomial (4.5) is needed,

$$\begin{aligned} \Lambda &= (1 + \alpha x + \alpha^6 x^2)(1 + \alpha^6 x + \alpha^5 x^2 + \alpha x^3 + \alpha x^4) \bmod x^5 \\ &= (1 + \alpha^5 x + \alpha^3 x^2) \bmod x^5, \end{aligned}$$

It is important to note that the Euclidean algorithm computes *simultaneously* $\sigma(x)$ and $\Lambda(x)$, as $\sigma(x) = b_{j_{\text{last}}}(x)$ and $\Lambda(x) = r_{j_{\text{last}}}(x)$. To verify this note that

$$r_3(x) = \alpha^3 + \alpha x + \alpha^6 x^2 = \alpha^3(1 + \alpha^5 x + \alpha^3 x^2) = \alpha^3 \Lambda(x).$$

With the error locations determined, the error values from Equation (4.4) are

$$\begin{aligned} e_2 &= (\alpha^2)^2(1 + \alpha^5 \alpha^{-2} + \alpha^3 \alpha^{-4})\alpha^{-1} = \alpha, \\ e_4 &= (\alpha^4)^2(1 + \alpha^5 \alpha^{-4} + \alpha^3 \alpha^{-8})\alpha^{-1} = \alpha^5. \end{aligned}$$

Therefore, $\bar{e}(x) = \alpha x^2 + \alpha^5 x^4$ and the decoded word is

$$\hat{c}(x) = \bar{r}(x) + \bar{e}(x) = 0.$$

The two errors have been corrected.

Note that the constant β is the same for both polynomials found by application of the extended Euclidean algorithm. The EA finds $\beta \cdot \sigma(x)$ and $\beta \cdot \Lambda(x)$, for some nonzero constant $\beta \in GF(2^m)$. Nevertheless, both error locator and error evaluator polynomials have *the same roots* as those obtained by the PGZ or BMA algorithms, and thus the error values obtained are the same.

In most of the computer programs to simulate encoding and decoding procedures of RS codes on the ECC web site, the following equivalent method of finding the error values is used [LC]. Let

$$z(x) = 1 + (S_1 + \sigma_1)x + (S_2 + \sigma_1 S_1 + \sigma_2)x^2 + \cdots + (s_\nu + \sigma_1 S_{\nu-1} + \cdots + \sigma_\nu)x^\nu. \quad (4.6)$$

Then the error value is computed as [Ber1]

$$e_{j_\ell} = \frac{(\alpha^{j_\ell})^{1-b} z(\alpha^{j_\ell})}{\prod_{\substack{i=1 \\ i \neq \ell}}^{\nu} (1 + \alpha^{j_i - j_\ell})}, \quad (4.7)$$

where $1 \leq \ell \leq \nu$.

Yet another alternative to Forney algorithm, for small values of t_d, is to determine the error values directly as follows. For $1 \leq \ell \leq \nu$, the error values e_{j_ℓ} are related to the syndromes S_i by the set of linear equations:

$$S_i = \bar{e}(\alpha^{b+i-1}) = \sum_{\ell=1}^{\nu} e_{j_\ell} \alpha^{(b+i-1)j_\ell}. \quad (4.8)$$

where $1 \leq i \leq 2t_d$.

Each $\nu \times \nu$ submatrix formed by the (known) terms $\alpha^{(b+i-1)j_\ell}$ forms a Vandermonde matrix. After all the ν error locations j_ℓ are known, any set of ν equations of the form (4.8) can be used to find the error values. In particular, choosing the first ν syndromes:

$$\begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_\nu \end{pmatrix} = \begin{pmatrix} (\alpha^{j_1})^b & (\alpha^{j_2})^b & \cdots & (\alpha^{j_\nu})^b \\ (\alpha^{j_1})^{b+1} & (\alpha^{j_2})^{b+1} & \cdots & (\alpha^{j_\nu})^{b+1} \\ \vdots & \vdots & \ddots & \vdots \\ (\alpha^{j_1})^{b+\nu-1} & (\alpha^{j_2})^{b+\nu-1} & \cdots & (\alpha^{j_\nu})^{b+\nu-1} \end{pmatrix} \begin{pmatrix} e_{j_1} \\ e_{j_2} \\ \vdots \\ e_{j_\nu} \end{pmatrix}, \quad (4.9)$$

is a system of linear equations that can be solved using $GF(2^m)$ arithmetic.

Example 42 Consider the same RS code and received polynomial in Examples 40 and 41. Then (4.9) gives:

$$\begin{pmatrix} \alpha^6 \\ \alpha^5 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \alpha^2 & \alpha^4 \end{pmatrix} \begin{pmatrix} e_2 \\ e_4 \end{pmatrix}.$$

The determinant of the 2×2 matrix is $\Delta = \alpha^4 + \alpha^2 = \alpha$. From this it follows that

$$e_2 = \alpha^{-1} \det \begin{pmatrix} \alpha^6 & 1 \\ \alpha^5 & \alpha^4 \end{pmatrix} = \alpha^6(\alpha^3 + \alpha^5) = \alpha^6 \cdot \alpha^2 = \alpha,$$

and

$$e_4 = \alpha^{-1}\det\begin{pmatrix} 1 & \alpha^6 \\ \alpha^2 & \alpha^5 \end{pmatrix} = \alpha^6(\alpha^5 + \alpha) = \alpha^6 \cdot \alpha^6 = \alpha^5,$$

which are the same error values as those obtained with Forney algorithm. Again, it is emphasized that this can only be done efficiently (and practically) for relatively small values of the error correcting capability, t_d, of the RS code.

4.3.1 Remarks on decoding algorithms

Unlike the BMA, in the EA all the syndromes are used in the first computation step. However, in terms of the number of $GF(2^m)$ operations, the BMA is generally more efficient than the EA. On the other hand, all the steps in the EA are identical, which translates into a more efficient hardware implementation. Also, the three decoding methods discussed here for (binary and nonbinary) BCH codes are examples of incomplete — or bounded distance — decoding. That is, they are able to detect situations in which the number of errors exceeds the capability of the code.

There are other approaches to decoding BCH codes, the most notable being the use of a discrete Fourier transform over $GF(2^m)$. This is covered extensively in [Blah], where the reader is referred to for details. Recently, Sudan [Sud] has introduced an algorithm that allows correction of errors beyond the minimum distance of the code. It applies to RS codes and more generally to AG codes. This algorithm produces a list of codewords (it is a list-decoding algorithm) and is based on interpolation and factorization of polynomials over $GF(2^m)$ and its extensions. Sudan algorithm was improved in [Gur].

4.3.2 Errors-and-erasures decoding

For the correction of erasures, the main change to the RS decoding procedures described above is that an *erasure locator polynomial* $\tau(x)$ needs to be introduced, defined as

$$\tau(x) = \prod_{\ell=1}^{\mu}(1 + y_{i_\ell}x),$$

where $y_{i_\ell} = \alpha^{i_\ell}$, for $1 \leq \ell \leq \mu$, denotes the position of an erasure.

By definition, the positions of the erasures are known. Therefore, *only the erasure values* need to be found. This can be done, as before, in the Forney algorithm step. In computing the syndromes of the received polynomial, it can be shown that any values of the erasures can be replaced, without any difference in the decoded word.

The decoding procedure is similar to the errors-only RS decoder, with the following exceptions. A modified syndrome polynomial, or *modified Forney syndrome*, is formed,

$$T(x) = S(x)\tau(x) + 1 \bmod x^{2t_d+1}. \tag{4.10}$$

The BMA algorithm can be applied to find $\sigma(x)$ with the following modifications:

1. The discrepancy is now defined as

$$d_i = T_{i+\mu+1} + \sum_{j=1}^{\ell_i} \sigma_j^{(i)} S_{i+\mu+1-i}, \tag{4.11}$$

with $d_0 = T_{\mu+1}$.

2. The algorithm finishes when the following stopping condition is met:

$$i \geq l_{i+1} + t_d - 1 - \mu/2.$$

After $\sigma(x)$ is obtained, a *modified errors-and-erasure evaluator*, or errata evaluator, is computed as $\omega(x)$,

$$\omega(x) = [1 + T(x)]\,\sigma(x) \bmod x^{2t_d+1}. \tag{4.12}$$

In addition, the following *errata locator polynomial* is computed,

$$\phi(x) = \tau(x)\sigma(x). \tag{4.13}$$

The resulting errata evaluation, or *modified Forney algorithm*, is given by

$$e_{j_\ell} = \frac{(\alpha^{j_\ell})^{2-b}\omega(\alpha^{-j_\ell})}{\phi'(\alpha^{-j_\ell})}, \tag{4.14}$$

$1 \leq \ell \leq \nu$, for the error values, and

$$f_{i_\ell} = \frac{(y_{i_\ell})^{2-b}\omega(y_{i_\ell}^{-1})}{\phi'(y_{i_\ell}^{-1})}, \tag{4.15}$$

$1 \leq \ell \leq \mu$, for the erasure values.

For errors-and-erasures decoding, the Euclidean algorithm can also be applied to the modified syndrome polynomial $T(x)$, using $1 + T(x)$ instead of $S(x)$ as in errors-only decoding. That is, the inital conditions are $r_0(x) = x^{2t_d+1}$ and $r_1(x) = 1 + T(x)$. The algorithm stops when $\deg[r_j(x)] \leq \lfloor (d - 1 + \mu)/2 \rfloor$, with $\omega(x) = r_j(x)$ and $\sigma(x) = b_j(x)$.

Example 43 Let C be an RS $(15, 9, 7)$ code over $GF(2^4)$ with zeros $\{\alpha, \alpha^2, \cdots, \alpha^6\}$, where α is a primitive element satisfying $p(\alpha) = \alpha^4 + \alpha^3 + 1 = 0$. As a reference for this example, a table of elements of $GF(2^4)$ as powers of a primitive element α, with $\alpha^4 + \alpha^3 + 1 = 0$, is shown below.

Table of elements of $GF(2^4)$, $p(x) = x^4 + x^3 + 1$.

Power	Vector
0	0000
1	0001
α	0010
α^2	0100
α^3	1000
α^4	1001
α^5	1011
α^6	1111
α^7	0111
α^8	1110
α^9	0101
α^{10}	1010
α^{11}	1101
α^{12}	0011
α^{13}	0110
α^{14}	1100

The generator polynomial of C is

$$\bar{g}(x) = \prod_{i=1}^{6}(x+\alpha^i) = x^6 + \alpha^{12}x^5 + x^4 + \alpha^2 x^3 + \alpha^7 x^2 + \alpha^{11}x + \alpha^6.$$

Suppose that the polynomial associated with a codeword $\bar{v}$ is

$$\begin{aligned}\bar{v}(x) &= \alpha^5 + \alpha^3 x + \alpha^{13}x^2 + \alpha x^3 + \alpha^7 x^4 + \alpha^4 x^5 + \alpha x^6 + \alpha^4 x^7 + \alpha^6 x^8 \\ &\quad + \alpha^3 x^{10} + \alpha^5 x^{11} + \alpha^6 x^{12} + \alpha^{13}x^{13} + \alpha^{10}x^{14}.\end{aligned}$$

Let the received polynomial be

$$\begin{aligned}\bar{r}(x) &= \alpha^7 + \alpha^3 x + \alpha^{13}x^2 + \alpha^{14}x^3 + \alpha^7 x^4 + \alpha x^5 + \alpha x^6 + \alpha^4 x^7 + \alpha^6 x^8 \\ &\quad + \alpha^3 x^{10} + \alpha^5 x^{11} + \alpha^{11}x^{12} + \alpha^{13}x^{13} + \alpha^{10}x^{14}.\end{aligned}$$

Assume that, aided by side information from the receiver, it is determined that the values in positions α^0 and α^5 are unreliable, and thus declared as erasures.

Note that $\bar{e}(x) = \alpha^{14} + \alpha^8 x^3 + \alpha^5 x^5 + \alpha x^{12}$ is the polynomial associated with the errata.[1]

After reception, besides $\bar{r}(x)$, the decoder knows that $\mu = 2$ erasures have occurred in positions α^0 and α^5. Therefore, it computes the erasure locator polynomial

$$\tau(x) = (1+x)(1+\alpha^5 x) = 1 + \alpha^{10}x + \alpha^5 x^2. \tag{4.16}$$

The syndromes of $\bar{r}(x)$ are computed as: $S_1 = \bar{r}(\alpha) = \alpha^{11}$, $S_2 = \bar{r}(\alpha^2) = \alpha^5$, $S_3 = \bar{r}(\alpha^3) = \alpha^2$, $S_4 = \bar{r}(\alpha^4) = \alpha^2$, $S_5 = \bar{r}(\alpha^5) = 1$ and $S_6 = \bar{r}(\alpha^6) = \alpha^{14}$. Accordingly,

$$S(x) = 1 + \alpha^{11}x + \alpha^5 x^2 + \alpha^2 x^3 + \alpha^2 x^4 + \alpha^5 + \alpha^{14}x^6. \tag{4.17}$$

The Forney syndrome polynomial (4.10) becomes

$$\begin{aligned}T(x) &= S(x)\tau(x) + 1 \bmod x^{2t_d+1} \\ &= (1 + \alpha^{11}x + \alpha^5 x^2 + \alpha^2 x^3 + \alpha^2 x^4)(1 + \alpha^{10}x + \alpha^5 x^2) + 1 \bmod x^7 \\ &= \alpha^7 x + \alpha^6 x^2 + \alpha^7 x^3 + \alpha^{11}x^4 + \alpha^9 x^5 + x^6,\end{aligned}$$

Berlekamp-Massey algorithm for errors-and-erasures correction

- **Iteration 0**: Initialize. $\sigma^{(-1)}(x) = 1$, $\ell_{-1} = 0$, $d_{-1} = 1$, $\sigma^{(0)}(x) = 1$, $\ell_0 = 0$, and $d_0 = T_3 = \alpha^7$.
- **Iteration 1:** $i = 0$, $d_0 = \alpha^7 \neq 0$. $m = -1$ maximizes $(-1+0) = -1$ for $d_{-1} \neq 0$.

$$\begin{aligned}&\sigma^{(1)}(x) = \sigma^{(0)}(x) + d_0 d_{-1}^{-1} x^{(0-(-1))}\sigma^{(-1)}(x) = 1 + \alpha^7 x, \\ &\ell_1 = \max\{\ell_0, \ell_{-1} + 0 - (-1)\} = 1, \\ &i \geq \ell_1 + 1 \ \text{?} \ \text{No}: \\ &d_1 = T_4 + T_3\sigma_1^{(1)} = \alpha^{11} + \alpha^7\alpha^7 = 1.\end{aligned}$$

[1] The decoder obviously does not know this, except for the positions of the erasures. This polynomial is used as a reference against which the correctness of the decoding results can be verified.

- **Iteration 2:** $i = 1$, $m = 0$ maximizes $(0 - 0) = 0$ for $d_0 \neq 0$.

$$\sigma^{(2)}(x) = \sigma^{(1)}(x) + d_1 d_0^{-1} \sigma^{(0)}(x) = 1 + \alpha^4 x.$$
$$\ell_2 = \max\{1, 0 + 1 - 0\} = 1,$$
$$i \geq 2 \ ? \text{ No :}$$
$$d_2 = T_5 + T_4 \sigma_1^{(2)} = \alpha^9 + \alpha^{11}\alpha^4 = \alpha^2.$$

- **Iteration 2:** $i = 2$, $m = 0$ maximizes $(0 - 0) = 0$ for $d_0 \neq 0$.

$$\sigma^{(3)}(x) = \sigma^{(2)}(x) + d_2 d_0^{-1} \sigma^{(0)}(x) = 1 + \alpha^4 x + \alpha^{10} x^2.$$
$$\ell_3 = \max\{1, 0 + 2 - 0\} = 2,$$
$$i \geq 3 \ ? \text{ No :}$$
$$d_3 = T_6 + T_5 \sigma_1^{(3)} + T_4 \sigma_2^{(3)} = 1 + \alpha^9\alpha^4 + \alpha^{11}\alpha^{10} = \alpha^3.$$

- **Iteration 3:** $i = 3$, $m = 2$ maximizes $(2 - 1) = 1$ for $d_2 \neq 0$.

$$\sigma^{(4)}(x) = \sigma^{(3)}(x) + d_3 d_2^{-1} \sigma^{(2)}(x) = 1 + \alpha^5 x + x^2.$$
$$\ell_4 = \max\{2, 1 + 3 - 2\} = 2,$$
$$i \geq 2 \text{ Yes : Algorithm ends.}$$

Therefore, $\sigma(x) = 1 + \alpha^5 x + x^2 = (1 + \alpha^3 x)(1 + \alpha^{12} x)$. For the last equality, recall that the inverses of the error positions are found, via Chien search, by evaluating $\sigma(x)$ at all the nonzero elements of $GF(2^4)$. This gives $\sigma(\alpha^{12}) = 0$ and $\sigma(\alpha^3) = 0$. As a result, the error positions are $\alpha^{-12} = \alpha^3$ and $\alpha^{-3} = \alpha^{12}$.

From Equation (4.12), the modified errors-and-erasures evaluator is

$$\begin{aligned} \omega(x) &= [1 + T(x)]\,\sigma(x) \bmod x^{2t_d+1} \\ &= (1 + \alpha^7 x + \alpha^6 x^2 + \alpha^7 x^3 + \alpha^{11} x^4 + \alpha^9 x^5 + x^6)(1 + \alpha^5 x + x^2) \bmod x^7 \\ &= 1 + \alpha^{14} x + \alpha^{11} x^2 + \alpha^{11} x^3 + x^4, \end{aligned} \tag{4.18}$$

and the errata locator polynomial

$$\phi(x) = \tau(x)\sigma(x) = (1 + \alpha^{10} x + \alpha^5 x^2)(1 + \alpha^5 x + x^2) = 1 + x + \alpha^5 x^2 + \alpha^5 x^4,$$

from which it follows that $\phi'(x) = 1$.

The values of the errors and erasures are found from (4.14) and (4.15), respectively,

$$\begin{aligned} e_3 &= \alpha^3(1 + \alpha^{11} + \alpha^5 + \alpha^2 + \alpha^3) = \alpha^3\alpha^5 = \alpha^8, \\ e_{12} &= \alpha^{12}(1 + \alpha^2 + \alpha^2 + \alpha^5 + \alpha^{12}) = \alpha^{12}\alpha^4 = \alpha, \\ f_0 &= (1 + \alpha^{14} + \alpha^{11} + \alpha^{11} + 1) = \alpha^{14}, \\ f_5 &= \alpha^5(1 + \alpha^9 + \alpha + \alpha^{11} + \alpha^{10}) = \alpha^5, \end{aligned}$$

from which it follows that the *errata polynomial* is

$$\bar{e}(x) = \alpha^{14} + \alpha^8 x^3 + \alpha^5 x^5 + \alpha x^{12}.$$

The decoded polynomial $\hat{v}(x) = \bar{r}(x) + \bar{e}(x)$ is identical to $\bar{v}(x)$. Two errors and two erasures have been corrected.

Direct solution of errata values

For small values of the minimum distance of an RS code, the erasure values may be obtained by solving a set of linear equations. Let $\bar{e}(x)$ be the error polynomial associated with an error pattern resulting from the presence of ν errors and μ erasures,

$$\bar{e}(x) = \sum_{\ell=1}^{\nu} e_{j_\ell} x^{j_\ell} + \sum_{\ell'=0}^{\mu} f_{j'_\ell} x^{j'_\ell}. \tag{4.19}$$

Then, the following set of linear equations, similar to (4.8), hold between the syndromes and the values of the errors and positions:

$$S_i = \bar{e}(\alpha^{b+i}) = \sum_{\ell-1}^{\nu} e_{j_\ell} \alpha^{(b+i)j_\ell} + \sum_{\ell'=1}^{\mu} f_{j'_\ell} \alpha^{(b+i)j'_\ell}, \tag{4.20}$$

where $1 \leq i \leq 2t_d$. As before, any set of $\nu + \mu \leq t_d$ equations can be used to solve the values of the errors and erasures.

Example 44 Direct solution of the errata values for the code in the previous example: After the Berlekamp-Massey algorithm and Chien search, the decoder knows that the error polynomial is of the form

$$\bar{e}(x) = f_0 + e_3 x^3 + f_5 x^5 + e_{12} x^{12}.$$

The errata values can be found by solving the set of linear equations given by (4.20), which can be put in the following matrix form,

$$\begin{pmatrix} 1 & \alpha^3 & \alpha^5 & \alpha^{12} \\ 1 & \alpha^6 & \alpha^{10} & \alpha^9 \\ 1 & \alpha^9 & 1 & \alpha^6 \\ 1 & \alpha^{12} & \alpha^5 & \alpha^5 \end{pmatrix} \begin{pmatrix} f_0 \\ e_3 \\ f_5 \\ e_{12} \end{pmatrix} = \begin{pmatrix} \alpha^{11} \\ \alpha^5 \\ \alpha^2 \\ \alpha^2 \end{pmatrix}. \tag{4.21}$$

It can be verified that $f_0 = \alpha^{14}$, $e_3 = \alpha^8$, $f_5 = \alpha^5$ and $e_{12} = \alpha$ are the solutions to (4.21). These are the same errata values as those computed before with the modified Forney algorithm.

4.4 Weight distribution

As mentioned before. RS codes are also MDS codes. The weight distribution of MDS codes can be computed exactly with a closed form expression [MS, LC]:

The number of codewords of weight i in an (n, k, d) MDS code over $GF(q)$ is

$$A_i = \binom{n}{i}(q-1)\sum_{j=0}^{i-d}(-1)^j \binom{i-1}{j} q^{i-j-d}. \tag{4.22}$$

For error performance evaluation of an RS (n, k, d) code over $GF(2^m)$, the following upper bound on the probability of a bit error $P_b(C)$ for an RS decoder[2] is simple to compute and

[2] It should be noted that the bound is tight only for *bounded distance* decoders, such as those based on BMA, EA or PGZ.

relatively good,

$$P_b(C) \leq \frac{2^{m-1}}{2^m - 1} \sum_{i=t_d+1}^{n} \frac{i + t_d}{n} \binom{n}{i} P_s^i (1 - P_s)^{n-i}, \tag{4.23}$$

where P_s denotes the probability of a *symbol error* at the input of the RS decoder,

$$P_S = 1 - (1 - p)^m,$$

and p denotes the probability of a *bit error* at the input of the RS decoder. The probability of a word error can be upper bounded by (1.31),

$$P_e(C) < 1 - \sum_{i=0}^{t_d} \binom{n}{i} P_s^i (1 - P_s)^{n-i}. \tag{4.24}$$

5

Binary convolutional codes

First introduced by Elias [Eli2], binary convolutional codes are perhaps the most popular form of binary error correcting codes and have found numerous applications: Wireless communications (IMT-2000, GSM, IS-95), digital terrestrial and satellite communication and broadcasting systems, and space communication systems, just to cite a few. They were referred to in [Eli2] as *convolutional parity-check symbols codes*. Their most popular decoding method to date, the *Viterbi algorithm* [Vit1], also finds applications in combinatorially equivalent problems such as maximum likelihood sequence detection (e.g., equalization) and partial-response signaling (e.g., magnetic recording). Most recently, it has been shown that convolutional codes, when combined with interleaving in a concatenated scheme, can perform very close to the Shannon limit [BGT]. In this chapter, the basic properties and decoding procedures for binary convolutional codes are described.

5.1 Basic structure

A convolutional code is an error correcting code that processes information *serially*, or continuously, in short block lengths. A convolutional encoder has *memory*, in the sense that the output symbols depend not only on the input symbols, but also on previous inputs and/or outputs. In other words, the encoder is a *sequential circuit* or a finite state machine. The *state* of the encoder is defined as the contents of the memory. In the computer programs that implement the Viterbi algorithm and other decoding procedures that involve a trellis, found on the ECC web site, a state transition table, indicating the relation between the input, the previous and current state, and the current output, is employed.

A convolutional code consists of the set of all binary sequences produced by a convolutional encoder. In theory, these sequences have infinite duration. In practice, the state of the convolutional code is periodically forced to a known state and code sequences produced in a block-wise manner.

Example 45 Consider the convolutional encoder depicted in Figure 29. For analysis and decoding purposes, this encoder can be described by Table 2.

Note that the *coding rate* of the convolutional encoder is equal to $1/2$, because two output coded bits are produced for every input information bit.

In general, a rate-k/n convolutional encoder has k shift registers, one per input information bit, and n output coded bits which are given by linear combinations (over the binary field, i.e., with exclusive-or gates) of the contents of the registers and the input information bits.

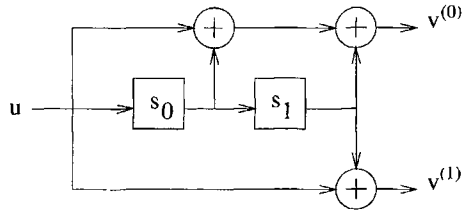


Figure 29 An encoder of a rate-1/2 memory-2 convolutional encoder.

Table 2 Input bits, state transitions and output bits.

Initial state $s_0[i]s_1[i]$	Information $u[i]$	Final state $s_0[i+1]s_1[i+1]$	Outputs $v^{(0)}[i]v^{(1)}[i]$
00	0	00	00
00	1	10	11
01	0	00	11
01	1	10	00
10	0	01	10
10	1	11	01
11	0	01	01
11	1	11	10

For simplicity of exposition, and for practical purposes, only rate-$1/n$ binary convolutional codes are considered in the remainder of the chapter. One reason for doing this is that these are the binary codes most widely used. A technique known as *puncturing* can be applied that results in convolutional encoders of higher rate. Puncturing is discussed in Section 5.5.

The total length of the shift registers in the encoder is referred to as the *memory*. For the discussion in this chapter, *states labels* are integers I associated with the binary representation of the memory contents, as $I = \sum_{j=0}^{m-1} s_j[i]2^{m-1-j}$.

The *constraint length*, which for rate-$1/2$ encoders equals $K = m + 1$, is defined as the number of inputs $(u[i], u[i-1], ..., u[i-m])$ that affect the outputs $(v^{(0)}[i], ..., v^{(n-1)}[i])$ at time i. For the encoder in Figure 29, $K = 3$. An encoder with m memory elements will be referred to as a memory-m encoder. A *memory-m rate-$1/n$* convolutional encoder can be represented by a *state diagram*. There are 2^m states in the diagram. Because there is only one information bit, two branches enter and leave each state and are labeled by $u[i]/v^{(0)}[i] \cdots v^{(n-1)}[i]$.

Example 46 A state diagram of the memory-2 rate-$1/2$ convolutional code of example 45 is shown in Figure 30.

The encoder of a memory-m rate-$1/n$ binary convolutional code can also be considered a *discrete linear time-invariant*[1] system. Therefore, the impulse response, i.e., the output of the encoder when the input is an "impulse" $\bar{u} = (1000...00...)$, completely specifies the code.

Convention: In writing a sequence, the leftmost symbol is meant to be the first symbol that goes into the channel.

[1] Note, however, that *time-varying* convolutional codes can be defined and are studied in [Joh].

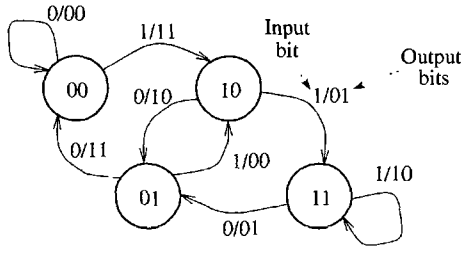


Figure 30 State diagram of a memory-2 rate-1/2 convolutional encoder.

There are n impulse responses of a convolutional encoder of rate $1/n$, one for each output $\bar{v}^{(j)}$, for $j = 0, 1, \cdots, n-1$. As the impulse goes across the memory elements of the encoder, it "picks up" the connections between the memory elements and the outputs. Evidently, this is a finite-impulse-response (FIR) system.

Let $\bar{g}_0$, ... $\bar{g}_{n-1}$ denote the impulse responses of a rate-$1/n$ convolutional encoder. These are also called the *generator sequences* — or generators — of the code, based on the observation above, and indicate the actual physical connections of the encoder.

Example 47 Continuing with the memory-2 rate-1/2 convolutional code of Example 45, the encoder in Figure 29 produces $\bar{g}_0 = (1, 1, 1)$ and $\bar{g}_1 = (1, 0, 1)$, as shown in Figure 31.

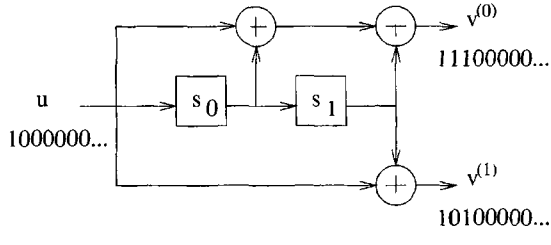


Figure 31 Generator sequences of a memory-2 rate-1/2 convolutional encoder.

Note that the generator sequences are equal to zero after K bits. Therefore, it suffices to consider vectors of this length. To stress the dynamic nature of the encoder, the indeterminate D (for "delay") is employed and the generators written as polynomials in D, $\bar{g}_0(D)$, ... $\bar{g}_{n-1}(D)$.

Example 48 For the encoder in Figure 29,

$$\bar{g}_0(D) = 1 + D + D^2 \quad \text{and} \quad \bar{g}_1(D) = 1 + D^2.$$

In general, when referring to a convolutional code, the generators are expressed as $(\bar{g}_0, \cdots, \bar{g}_{n-1})$ and written in octal form. For the encoder of Example 48 the generators are $(7, 5)$. The most widely used convolutional encoder to date is a memory-6 rate-1/2 encoder with generators $(171, 133)$. This is the so-called *NASA standard* code, as it was first used in the Voyager space missions. It is also used in many digital communications standards, as well as in the CCSD standard.

As a result of the dynamical structure of a convolutional encoder, the state diagram can be represented as it evolves in time with a *trellis diagram*. A trellis diagram is constructed by placing the state diagram of the code at each time interval with branches connecting states between time i and time $i+1$, in correspondence with the encoder table. The branches of the trellis are labeled in the same way as the state diagram.

Convention: When there is no dichotomy, the input information bit does not need to appear explicitly in the branch label. For FIR encoders, the information bit u can be inferred directly from the state transition[2]:

$$(s_0 s_1 \cdots s_{m-1}) \rightarrow (u s_0 \cdots s_{m-2}).$$

It should be emphasized that, in the case of a recursive systematic convolutional (IIR) encoder, this is not generally the case, as is shown in Section 5.1.1.

Example 49 For the example rate-$1/2$ memory-2 convolutional encoder, the trellis structure is shown in Figure 32. Note that state transitions (branches) form a state $s[i]$ to a (possibly different) state $s'[i+1]$, that caused by the input bit $u[i]=0$, are the upper branches.

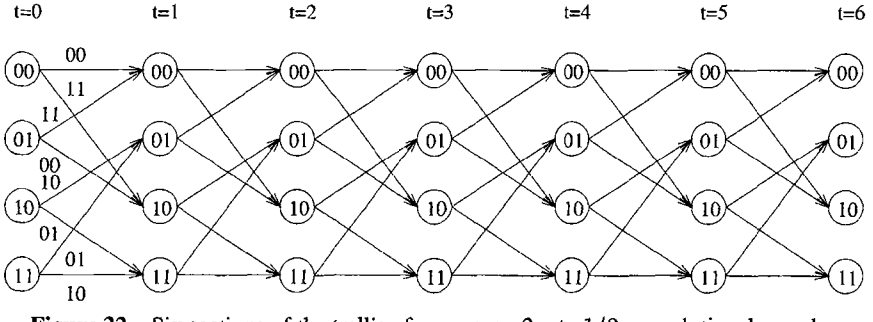


Figure 32 Six sections of the trellis of a memory-2 rate-$1/2$ convolutional encoder.

Example 50 Consider the encoder in Figure 29 and the input sequence $\bar{u} = (110100)$. Then, the corresponding output sequence can be obtained either directly from the encoder table, Table 2, or as a path in the trellis, as shown in Figure 33. It follows that the corresponding coded sequence is $\bar{v} = (11, 01, 01, 00, 10, 11)$.

A convolutional encoder is a linear time-invariant system, with impulse responses given by the code generators, $\bar{g}_0(D), \bar{g}_1(D), \cdots, \bar{g}_n(D)$, where

$$\bar{g}_j(D) = g_j[0] + g_j[1]D + g_j[2]D^2 + \cdots + g_j[m]D^m, \tag{5.1}$$

for $0 \leq j < n$. In terms of the generators of a code, the output sequences can be expressed as

$$v^{(j)}[i] = \sum_{\ell=0}^{m} u[i-\ell] g_j[\ell], \tag{5.2}$$

[2] This simple observation is the basis of using a *traceback memory* in a Viterbi decoder, to recover the information bits from the decoded sequence.

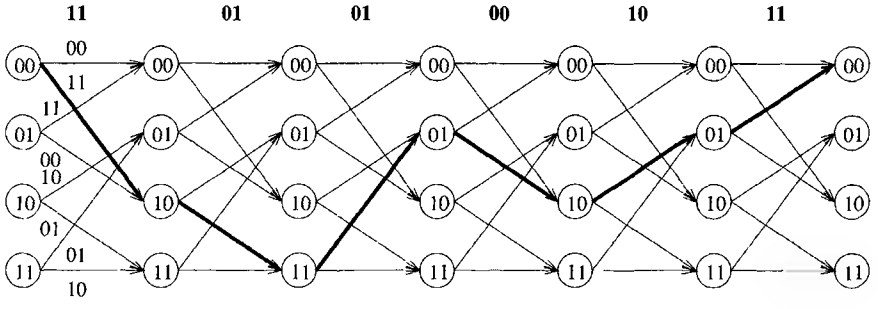


Figure 33 A path in the trellis of a memory-2 rate-1/2 convolutional encoder.

$0 \leq j < n$. The output sequences $\bar{v}^{(j)}(D)$, $0 \leq j < n$, as expected, are equal to the *discrete convolution* between the input sequence $\bar{u}(D)$ and the code generators $\bar{g}_0(D)$, $\bar{g}_1(D), \cdots, \bar{g}_{n-1}(D)$. From this fact comes the name – "convolutional"– of the encoder [Eli2].

Observe now that Equation (5.2) above can be written as a matrix multiplication

$$\bar{v} = \bar{u}G, \tag{5.3}$$

where G is a *generator matrix* of the convolutional code. In particular, for a rate-$1/2$ memory-m convolutional code,

$$G = \begin{pmatrix} g_0[0]g_1[0] & \cdots & g_0[m]g_1[m] & & \\ & g_0[0]g_1[0] & \cdots & g_0[m]g_1[m] & \\ & & g_0[0]g_1[0] & \cdots & g_0[m]g_1[m] \\ & & \ddots & & \ddots \end{pmatrix}, \tag{5.4}$$

where the blank entries represent zeros. Generalization to any rate $1/n$ encoder is straightforward.

Example 51 For the memory-2 rate-$1/2$ encoder in Figure 29,

$$G = \begin{pmatrix} 11 & 10 & 11 & & & & \\ & 11 & 10 & 11 & & & \\ & & 11 & 10 & 11 & & \\ & & & 11 & 10 & 11 & \\ & & & & \ddots & & \ddots \end{pmatrix}.$$

Let $u = (110100)$. Then it follows from (5.3) that $\bar{v} = \bar{u}G = (11, 01, 01, 00, 10, 11)$, which is the same output as in Example 50.

Let $v(D) = v^{(0)}(D) + Dv^{(1)}(D) + \cdots + D^{n-1}v^{(n-1)}(D)$. Then the relationship between input and output sequences can be written as

$$\bar{v}(D) = \bar{u}(D)G(D), \tag{5.5}$$

where generators of a rate-$1/n$ convolutional code are arranged in a matrix, referred to as a *polynomial generator matrix*,

$$G(D) = \begin{pmatrix} \bar{g}_0(D) & \bar{g}_1(D) & \cdots & \bar{g}_{n-1}(D) \end{pmatrix}. \tag{5.6}$$

5.1.1 *Recursive systematic convolutional codes*

With the use of (5.6), a memory-m rate-$1/n$ *recursive systematic convolutional code*, or RSC code, has a generator matrix of the form

$$G'(D) = \left(I \quad \frac{\bar{g}_1(D)}{\bar{g}_0(D)} \quad \cdots \quad \frac{\bar{g}_1(D)}{\bar{g}_0(D)} \right). \tag{5.7}$$

As a short notation of this generator matrix, the *generators* $(1, \bar{g}_1/\bar{g}_0, \cdots, \bar{g}_{n-1}/\bar{g}_0)$ can be specified. Division and multiplication of the right-hand side of (5.5) by $\bar{g}_0(D)$ give, together with (5.6),

$$\bar{v}(D) = \bar{u}'(D)G'(D), \tag{5.8}$$

where $\bar{u}'(D) = \bar{g}_0(D)\bar{u}(D)$. This shows that both the nonsystematic encoder and the systematic encoder produce the same coded sequences, but that the corresponding information sequences are scrambled by $\bar{g}_0(D)$ in the RSC code.

A systematic encoder is also an example of a discrete-time linear time-invariant system. Because the generator matrix contains rational functions, the encoder is an *infinite-impulse-response (IIR)* linear system, as opposed to a non-systematic encoder which is a *finite impulse response (FIR)* linear system. The preferred form of expressing the encoder circuit is the *controller canonical form* [For4]. An encoder for an RSC code, consists of a shift registers, a circuit[3] for dividing by $\bar{g}_0(D)$ and $n-1$ circuits for multiplying by $\bar{g}_1(D), \cdots, g_{n-1}(D)$.

Example 52 A rate-$1/2$ memory-2 binary RSC encoder with generators $(1, 5/7)$ is shown in Figure 34. The trellis structure of this code is the same as the nonsystematic code with generators $(7, 5)$. However the input/output bits per transition are different. The state diagram of this code is shown in Figure 35. Compare with Figure 30 on page 75. Among other things, this difference in input/ouput mapping means that the all-zero sequence may not necessarily bring the state back to $s_0 s_1 = 00$. This is evident from Figure 35. In particular, the impulse response of this encoder is $(11, 01, 01, 00, 01, 01, 00, 01, 01, 00, \cdots)$.

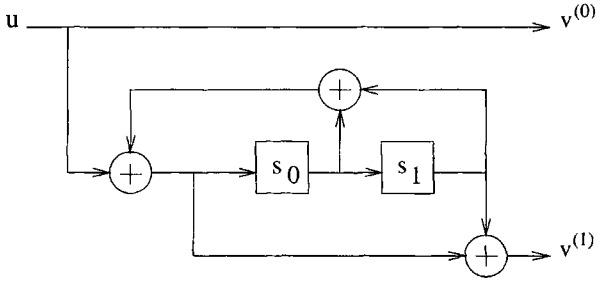


Figure 34 Encoder of a memory-2 rate-$1/2$ recursive systematic convolutional encoder.

[3] The general structure of circuits that multiply and divide by two polynomials can be found, among others, in [PW], Figure 7.8, or [LC], Sec. 4.7.

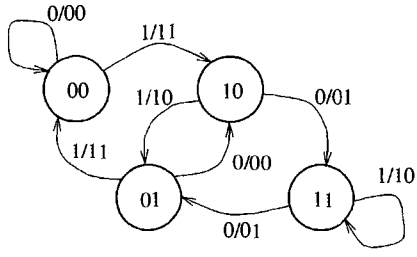


Figure 35 State diagram of a memory-2 rate-1/2 recursive systematic convolutional encoder.

5.1.2 Free distance

The *free distance* d_f of a convolutional code is the smallest distance between any two distinct code sequences. The length of the sequences has to be sufficiently large, much larger than the constraint length of the code. The free distance of a convolutional code can be obtained from its weight enumerator polynomial, as described in Section 5.3. There are other distances associated with a convolutional code, when the length of the sequence is of the order of the constraint length, but these are not relevant for the discussion in this book. More details on the structure of a general rate-k/n convolutional encoder can be found in references [LC] and [Joh].

5.2 Connections with block codes

There is a connection between convolutional codes and block codes, as it is evident from the above description of convolutional encoders. As indicated previously, it is customary for the information sequences input to a convolutional encoder to be broken into blocks of *finite length* (e.g. a few thousands of bits). Generally, a fixed sequence of length m is appended at the end of each information sequence. This sequence is typically a *unique word* that serves to synchronize the receiver and forces the convolutional encoder to return to a known state.

In the following, for simplicity of exposition, let C be a nonsystematic (FIR) convolutional code C with free distance d_f, obtained from a memory-m rate-$1/n$ convolutional encoder. Similar arguments apply to RSC codes.

5.2.1 Zero-tail construction

By appending a "tail" of m zeros to an information vector $\bar{u}$ of length $(K - m)$, all paths in the trellis corresponding to 2^{K-m} codewords merge at the all-zero state. The result is a linear block $(nK, (K - m), d_{\mathrm{ZT}})$ code, denoted by C_{ZT}. If K is large enough, then the rate of C_{ZT} approaches the rate of C. As long as $K > m$ holds, with K sufficiently large, even though the code sequences are restricted to end at the all-zero state, the minimum distance of C_{ZT} satisfies $d_{\mathrm{ZT}} = d_f$.

Example 53 Let C be the convolutional code obtained from the rate-1/2 memory-2 encoder in Figure 29, of free distance[4] $d_f = 5$. Assume that message vectors of length $K = 3$ bits are encoded, each followed by $m = 2$ zeros. Then the zero-tail construction results in a binary

[4] That this is the value of d_f will be shown in Section 5.3.

linear block $(10, 3, 5)$ code C_{ZT} with generator matrix,

$$G = \begin{pmatrix} 11 & 10 & 11 & 00 & 00 \\ 00 & 11 & 10 & 11 & 00 \\ 00 & 00 & 11 & 10 & 11 \end{pmatrix}.$$

The weight distribution sequence of this code is $A(x) = 1 + 3x^5 + 3x^6 + x^7$, and $d_{ZT} = 5$. The code rate of C_{ZT} is 0.3 which is less than the rate $k/n = 1/2$ of C.

5.2.2 *Direct-truncation construction*

In this method, the codewords of a linear block (nK, K, d_{DT}) code C_{DT} associated with a rate-k/n convolutional encoder are all code sequences associated with paths in the trellis that start in the all-zero state and, after K bits are fed into the encoder, can end at any state. The rate of C_{DT} is the same as that of the convolutional encoder. However, the minimum distance of the resulting block code is reduced and $d_{DT} < d_f$.

Example 54 Consider the rate-$1/2$ memory-2 encoder in Figure 29. Information vectors of length $K = 3$ information bits are encoded. Then the direct truncation construction gives a binary linear block $(6, 3, d_{DT})$ code with generator matrix,

$$G = \begin{pmatrix} 11 & 10 & 11 \\ 00 & 11 & 10 \\ 00 & 00 & 11 \end{pmatrix}.$$

The weight distribution sequence of this code is $A(x) = 1 + x^2 + 3x^3 + 2x^4 + x^5$, and $d_{DT} = 2 < d_f$.

5.2.3 *Tail-biting construction*

The codewords of the tail-biting block code C_{TB} are those code sequences associated with paths in the trellis that start from a state equal to the last m bits of an information vector of K data bits. This ensures that all codewords in C_{TB} begin and end *at the same state*. The rate of the resulting block $(2K, K, d_{TB})$ code is the same as that of the convolutional encoder. However, unless $K > m$ and K is sufficiently large, the minimum distance $d_{TB} \leq d_f$.

Example 55 Consider the rate-$1/2$ memory-2 encoder in Figure 29. Assume that a block of $K = 5$ information bits is encoded. Then the tail-biting construction gives a binary linear block $(10, 5, d_{TB})$ code with generator matrix,

$$G = \begin{pmatrix} 11 & 10 & 11 & 00 & 00 \\ 00 & 11 & 10 & 11 & 00 \\ 00 & 00 & 11 & 10 & 11 \\ 11 & 00 & 00 & 11 & 10 \\ 10 & 11 & 00 & 00 & 11 \end{pmatrix}.$$

The weight distribution sequence[5] is $A(x) = 1 + 5x^3 + 5x^4 + 6x^5 + 10x^6 + 5x^7$, and $d_{TB} = 3 < d_f$.

---

[5] This WDS was obtained with a program from the ECC web site that computes the weight distribution of a binary linear code from its generator matrix.

5.2.4 *Weight distributions*

In this section, a method is described to obtain the weight distribution of linear block codes derived from binary rate-$1/n$ convolutional codes by any of the constructions in the previous sections [WV]. Let $\Omega(x)$ be a $2^m \times 2^m$ *state transition matrix* with entries of the form

$$\Omega_{ij}(x) = \delta_{ij} x^{h_{ij}}, \tag{5.9}$$

where $\delta_{ij} = 1$, if and only if there is a transition from state i to state j, and $\delta_{ij} = 0$ otherwise; and h_{ij} is the Hamming weight of the corresponding output vector (of length n).

Example 56 For the convolutional encoder with the state diagram shown in Figure 30, the state transition matrix $\Omega(x)$ is given by

$$\Omega(x) = \begin{pmatrix} 1 & 0 & x^2 & 0 \\ x^2 & 0 & 1 & 0 \\ 0 & x & 0 & x \\ 0 & x & 0 & x \end{pmatrix}.$$

The weight distribution sequence of a binary linear block (n, k) code constructed from any of the previous methods can be obtained by symbolically raising matrix $\Omega(x)$ to the ℓ-th power, denoted $\Omega^{\ell}(x)$, and combining different terms.

The term $\Omega^{\ell}_{ij}(x)$ gives the weight distribution of trellis paths that start in state i and end at state j after ℓ time intervals. For the ZT construction, the value $\ell = k + m$ is used, while for both DT and TB constructions, $\ell = k$. The weight distribution sequences for each of the construction methods presented above can be computed as follows.

- **ZT construction**:

$$A(x) = \Omega^{k+m}_{00}(x).$$

- **DT construction**:

$$A(x) = \sum_{j=0}^{2^m} \Omega^{k}_{0j}(x).$$

- **TB construction**:

$$A(x) = \sum_{j=0}^{2^m} \Omega^{k}_{jj}(x).$$

Example 57 Consider again the memory-2 rate-1/2 convolutional encoder. Then

$$\Omega^3(x) = \begin{pmatrix} 1+x^5 & x^3+x^4 & x^2+x^3 & x^3+x^4 \\ x^2+x^3 & x^5+x^2 & x^4+x & x^5+x^2 \\ x^3+x^4 & x^2+x^3 & x^5+x^2 & x^2+x^3 \\ x^3+x^4 & x^2+x^3 & x^5+x^2 & x^2+x^3 \end{pmatrix}.$$

The weight distribution sequence of the code from the DT construction in Example 54 is obtained by adding the terms in the first row of $\Omega^3(x)$.

References on other methods of computing the weight distribution sequence of a linear block code derived from a convolutional code are [FLC], [CK] and [DFK1].

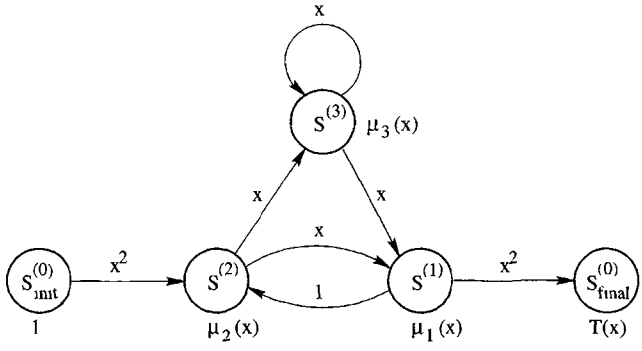


Figure 36 The modified state diagram of a memory-2 rate-1/2 convolutional code.

5.3 Weight enumeration and performance bounds

The performance of convolutional codes over memoryless channels can be estimated by using union bounds, such as those presented in Chapter 1 for block codes. It was Viterbi [Vit2] that first considered the performance of convolutional codes. See also [ViOm]. For a convolutional code, the *weight enumerating sequence* (WES) is defined as

$$T(x) = T_1 x + T_2 x^2 + \cdots + T_w x^w + \cdots, \tag{5.10}$$

where the coefficient T_w in $T(x)$ denotes the number of code sequences of weight w. Note that, in principle, the inputs to a convolutional code are infinite sequences. As a result, in general, the WES has an infinite number of terms. It is possible, however, to obtain closed-form expressions for the WES of convolutional codes, as shown below.

Notation: A state $(s_0 s_1 \cdots s_{m-1})$ of the encoder will be denoted by $S^{(I)}$, where $I = \sum_{i=0}^{m-1} s_i 2^{m-1-i}$.

A *modified state diagram* enables us to compute the WES. This diagram has the all-zero state split into an initial state $S_{\text{init}}^{(0)}$ and a final state $S_{\text{final}}^{(0)}$. The branches of the diagram are labeled by terms x^w, where w is the Hamming weight of the output sequence.

Example 58 Let C be the memory-2 rate-1/2 convolutional code with generators $(7, 5)$ that has been used throughout this chapter. Figure 36 shows the modified state diagram of C.

There are basically two methods to compute $T(x)$. In the first method, the modified state diagram is considered a signal flow graph. Then Mason's rule is applied to find $T(x)$. This is the method covered in most textbooks on error correcting codes. For details, the reader is referred to [LC].

Another method, originally developed by Viterbi [Vit2], consists of assigning intermediate variables $\mu_j(x)$ to each state $S^{(j)}$, $j = 1, 2, \cdots, 2^m - 1$. Each $\mu_j(x)$ can be expressed as a linear combination of weighted variables associated with states connected to $S(j)$,

$$\mu_j(x) = \sum_{\substack{k=1 \\ k \neq j}}^{2^m - 1} \mu_k(x) \delta_{kj} x^{h_{kj}}, \tag{5.11}$$

for all $k \in \{1, 2, \cdots, 2^m - 1\}$, $k \neq j$, and where δ_{ij} and h_{kj} are defined as in (5.9). The initial state $S_{\text{init}}^{(0)}$ is assigned a variable with value 1, while the WES $T(x)$ is assigned as a variable to the final state $S_{\text{final}}^{(0)}$,

This yields a system of linear equations that can be solved to find $\mu_1(x)$, from which, with the adopted notation to designate states,

$$T(x) = \mu_1(x) x^{h_{10}}. \tag{5.12}$$

Example 59 Consider the modified state diagram in Figure 36. The following set of equations is obtained

$$\begin{aligned} \mu_1(x) &= \mu_2(x)x + \mu_3(x)x \\ \mu_2(x) &= x^2 + \mu_1(x) \\ \mu_3(x) &= \mu_2(x)x + \mu_3(x)x \end{aligned} \tag{5.13}$$

and $T(x) = \mu_1(x)x^2$. Equations (5.13) can be written in matrix form as

$$\begin{pmatrix} 1 & -x & -x \\ -1 & 1 & 0 \\ 0 & -x & 1-x \end{pmatrix} \begin{pmatrix} \mu_1(x) \\ \mu_2(x) \\ \mu_3(x) \end{pmatrix} = \begin{pmatrix} 0 \\ x^2 \\ 0 \end{pmatrix}, \tag{5.14}$$

and solving for $\mu_1(x)$ gives

$$\mu_1(x) = \frac{x^3}{1-2x}.$$

It follows that

$$T(x) = \frac{x^5}{1-2x} = x^5 + 2x^6 + 4x^7 + 8x^8 + \cdots \tag{5.15}$$

This shows that the minimum free distance is $d_f = 5$ for the binary 4-state rate-1/2 convolutional code with generators $(7, 5)$.

More detailed information about the structure of a convolutional code is obtained by labeling the modified state diagram with terms of the form $x^w y^\ell z^m$, where w is the Hamming weight of the coded outputs, ℓ is the Hamming weight of the input vector (of length equal to k in general for a rate-k/n convolutional code), and m is the number of branches differing from the all-zero sequence. (See [LC], p. 302.) This results in a *complete weight enumerator sequence* (CWES), $T(x, y, z)$ which can be used to derive various bounds on the error performance of convolutional codes [Vit2], [LC], [Joh]. A weight enumerating sequence similar to the CWES was used to estimate the performance of turbo codes in [BM].

Example 60 Continuing with the example 4-state rate-1/2 convolutional code, a modified state diagram for the computation of the CWES is shown in Figure 37.

The equations are

$$\begin{pmatrix} 1 & -xz & -xz \\ -yz & 1 & 0 \\ 0 & -xyz & 1-xyz \end{pmatrix} \begin{pmatrix} \mu_1(x) \\ \mu_2(x) \\ \mu_3(x) \end{pmatrix} = \begin{pmatrix} 0 \\ x^2yz \\ 0 \end{pmatrix},$$

with $T(x, y, z) = x^2 z \mu_1(x)$. The solution is:

$$T(x, y, z) = \frac{x^5 y^3 z}{1 - xy(1+y)z} = x^4 y^3 z + x^3 y^4 (1+y) z^2 + x^7 y^5 (1+y)^2 z^3 + \cdots$$

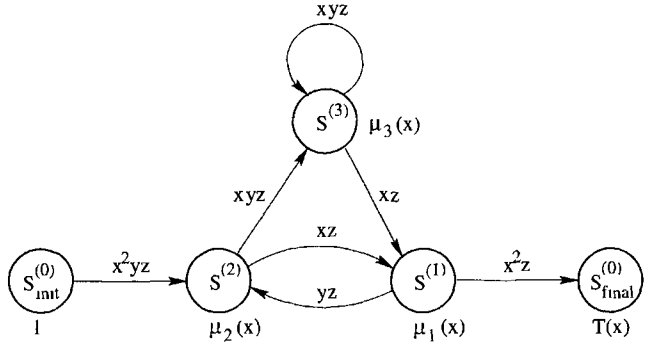


Figure 37 The modified state diagram of a memory-2 rate-1/2 convolutional code.

Bounds on the bit error probability of a binary convolutional code of rate k/n can be obtained with the aid of the CWES. For transmission over a BSC channel and binary transmission over an AWGN channel [ViOm], and maximum-likelihood decoding (MLD)[6], the following upper bounds hold, respectively,

$$P_b < \frac{1}{k} \left. \frac{\partial T(x,y,z)}{\partial y} \right|_{x=\sqrt{2p(1-p)},y=1,z=1}, \tag{5.16}$$

$$P_b < \frac{1}{k} \left. \frac{\partial T(x,y,z)}{\partial y} \right|_{x=e^{-E_s/N_0},y=1,z=1}. \tag{5.17}$$

Union bounds may be used to estimate the bit-error rate (BER) performance of a convolutional code. The reader should be aware, however, that the bounds (5.16) and (5.17) are quite loose. Fortunately, tighter (close to the actual BER performance) bounds exist at relatively mild channel conditions, i.e., low values of p for the BSC channel and high values of E_s/N_0 for the AWGN channel, and are given by the following [Joh]:

$$P_b < \frac{1}{k} \left(\frac{(1+x)}{2} \frac{\partial T(x,y,z)}{\partial y} + \frac{(1-x)}{2} \frac{\partial T(-x,y,z)}{\partial y} \right) \Bigg|_{x=\sqrt{2p(1-p)},y=1,z=1}, \tag{5.18}$$

$$P_b < \frac{1}{k} Q\left(\sqrt{2R\, d_f \frac{E_b}{N_0}} \right) e^{2R\, d_f E_b/N_0} \left. \frac{\partial T(x,y,z)}{\partial y} \right|_{x=e^{-E_s/N_0},y=1,z=1}. \tag{5.19}$$

Example 61 The union bound (5.18) on the probability of a bit error for the 4-state rate-1/2 convolutional code of example 60, with transmission over a BSC channel with crossover probability p and MLD, is plotted in Figure 38.

5.4 Decoding: Viterbi algorithm with Hamming metrics

The trellis of convolutional codes has a regular structure. Advantage can be taken from the repetitive pattern of the trellis in decoding. However, for linear block codes obtained

[6] An example of MLD is the Viterbi decoder, explained in the next section.

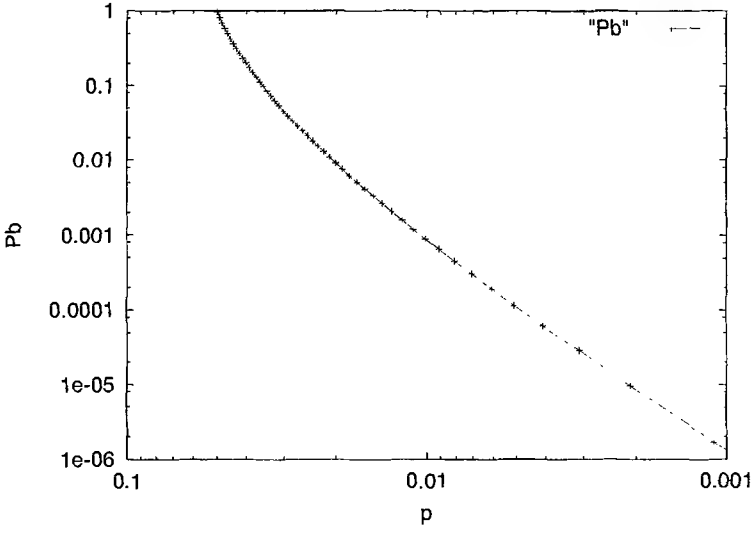


Figure 38 Union bound (5.18) on the BER of a memory-2 rate-1/2 convolutional code with $d_f = 5$. Transmission over BSC with crossover probability p and MLD.

from terminating convolutional codes, and long information sequences, maximum-likelihood decoding is simply too complex and inefficient to implement.

An efficient solution to the decoding problem is a dynamic programming algorithm known as the *Viterbi algorithm*, also known as the *Viterbi decoder* (VD). The VD is a *maximum likelihood decoder*, in the following sense. The VD finds the closest coded sequence $\bar{v}$ to the received sequence $\bar{r}$ by processing the sequences on an information bit-by-bit (branches of the trellis) basis. Instead of keeping a score of each possible coded sequence, *the Viterbi decoder tracks the states of the trellis.*

5.4.1 *Maximum likelihood decoding and metrics*

The *likelihood* of a received sequence $\bar{r}$ after transmission over a noisy channel, given that a coded sequence $\bar{v}$ is sent, is given by the conditional probability

$$P(\bar{r}|\bar{v}) = \prod_{i=0}^{n-1} P(r_i|v_i), \tag{5.20}$$

It is easy to show that for a BSC channel with parameter p,

$$P(\bar{r}|\bar{v}) = \prod_{i=0}^{n-1} (1-p) \left(\frac{p}{1-p}\right)^{d_H(r_i,v_i)}, \tag{5.21}$$

with $d_H(r_i, v_i) = 1$, if $r_i \neq v_i$, and $d_H(r_i, v_i) = 0$, if $r_i = v_i$. That is, $d_H(r_i, v_i) = 1$ is the *Hamming distance* between bits r_i and v_i. For an AWGN channel, the likelihood is given by

$$P(\bar{r}|\bar{v}) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{\pi N_0}} e^{-\frac{1}{N_0}(r_i - m(v_i))^2}, \tag{5.22}$$

where $m(\cdot)$ denotes a binary modulated signal. Here, m is defined as a *one-to-one mapping* between bits $\{0, 1\}$ and real numbers $\{-\sqrt{E}, +\sqrt{E}\}$.

A *maximum-likelihood decoder (MLD)* selects a coded sequence $\bar{v}'$ that *maximizes* (5.20). By taking the logarithm (any base) of (5.20), the following can be shown. For the BSC channel, an MLD is equivalent to choosing the code sequence that minimizes the *Hamming distance*

$$d_H(\bar{r}, \bar{v}) = \sum_{i=0}^{n-1} d_H(r_i, v_i), \tag{5.23}$$

Similarly, for the AWGN channel, it is the squared *Euclidean distance*

$$d_E(\bar{r}, \bar{v}) = \sum_{i=0}^{n-1} \left(\bar{r} - m(\bar{v})\right)^2, \tag{5.24}$$

that is minimized by the coded sequence selected by the MLD. In this section, a BSC channel with crossover error probability p is considered. The AWGN channel is covered in Chapter 7.

5.4.2 *The Viterbi algorithm*

Let $S_i^{(k)}$ denote a state in the trellis at stage i. Each state $S_i^{(k)}$ in the trellis is assigned a state metric, or simply a *metric*, $M(S_i^{(k)})$, and a *path* in the trellis, $\bar{y}^{(k)}$. A key observation in applying the Viterbi algorithm is:

> *At time i, the most likely paths per state $\bar{y}_i^{(k)}$ (the ones closest to the received sequence) will eventually coincide at some time $i - \ell$.*

In his paper, Viterbi [Vit1] indicated that the value of ℓ for memory-m rate-$1/2$ binary convolutional codes should be $\ell > 5m$. The VD operates within a range of L received n-tuples (output bits per state transition) known as the *decoding depth*. The value of L must be such that $L > \ell$.

In the following, the Viterbi algorithm applied to a memory-m rate-$1/n$ binary convolutional code is described, and its operation illustrated via a simple example. Some additional notation is needed: Let $\bar{v}[i] = (v_0[i]v_1[i] \cdots v_{n-1}[i])$ denote the coded bits in a branch (state transition), and let $\bar{r}[i] = (r_0[i]r_1[i] \cdots r_{n-1}[i])$ denote the output of the channel.

Basic decoding steps

Initialization

Set $i = 0$. Set metrics and paths

$$M(S_0^{(k)}) = 0, \qquad \bar{y}_0^{(k)} = () \quad \text{(empty)}.$$

The specific way in which the initialization of the paths is performed is irrelevant, as shown later. For the sake of clarity of presentation of the algorithm, it is assumed that the paths are represented as *lists* that are initialized to the empty list.

1. **Branch metric computation**
   At stage i, compute the partial *branch metrics*

$$BM_i^{(b)} = d_H(\bar{r}[i], \bar{v}[i]), \tag{5.25}$$

$b \triangleq \sum_{\ell=0}^{n-1} v_\ell[i] 2^{n-1-\ell}$, associated with the n outputs $\bar{v}[i]$ of every branch (or state transition) and the n received bits $\bar{r}[i]$.

2. **Add, compare and select (ACS)**
   For each state $S_i^{(k)}$, $k = 0, 1, \cdots, 2^m - 1$, and corresponding pair of incoming branches, from two precursor states $S_{i-1}^{(k_1)}$ and $S_{i-1}^{(k_2)}$, compare $M(S_{i-1}^{(k_1)}) + BM_i^{(b_1)}$ and $M(S_{i-1}^{(k_2)}) + BM_i^{(b_2)}$, where $b_j = \sum_{\ell=0}^{n-1} v_\ell[i] 2^{n-1-\ell}$, $i = 1, 2$. Select a *winning branch* giving the smallest path metric and update,

$$M(S_i^{(k)}) = \min\{M(S_{i-1}^{(k_1)}) + BM_i^{(b_1)}, M(S_{i-1}^{(k_2)}) + BM_i^{(b_2)}\}. \tag{5.26}$$

3. **Path memory update**
   For each state $S_i^{(k)}$, $k = 0, 1, \cdots, 2^m - 1$, update the *survivor paths* $\bar{y}^{(k)}$ as follows, with the output of the winning branch $\bar{v}_{k_j}$, $j \in \{1, 2\}$.

$$\bar{y}_i^{(k)} = (\bar{y}_{i-1}^{(k_j)}, \bar{v}_{k_j}), \tag{5.27}$$

4. **Decode symbols**
   If $i > L$, then output as the estimated coded sequence $\bar{y}_{i-L}^{(k')}$, where k' is the index of the state $S^{(k')}$ with the smallest metric. Set $i = i + 1$ and go to decoding step 1.

It should be stressed that this is not the only way to implement the Viterbi algorithm. The above procedure can be considered a *classical* algorithm. There are alternative implementations that, depending on the particular structure of the underlying convolutional encoder, may offer advantages (see, e.g., [FL2]). In addition, in the last step of the algorithm, symbol decoding can be applied to information bits directly. This is the form usually employed in the software implementations of Viterbi decoders available on the ECC web site. In hardware implementations, a method based on a *traceback memory* is favored that estimates the original information sequence, indirectly, based on state transitions. This technique is discussed later in the chapter.

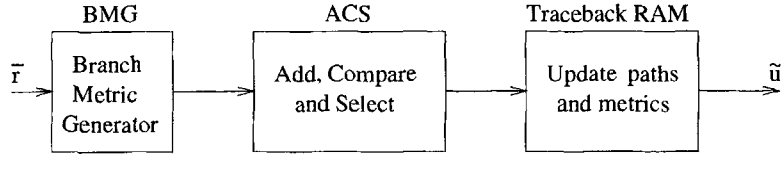


Figure 39 Block diagram of a Viterbi decoder.

Example 62 Consider again the memory-2 rate-1/2 convolutional encoder with generators $(7, 5)$. Note that $d_f = 5$ for this code. This example shows how a single error can be corrected. Suppose that $\bar{v} = (11, 01, 01, 00, 10, 11)$ is transmitted over a binary symmetric channel (BSC), and that $\bar{r} = (10, 01, 01, 00, 10, 11)$ is received (one error in the second position.) The operation of the Viterbi decoder is illustrated in Figures 40 to 45. The evolution of the metric values with respect to the decoding stages is shown in the following table:

State/Stage	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
$S_i^{(0)}$	0	1	1	1	1	2	1
$S_i^{(1)}$	0	0	0	1	2	1	3
$S_i^{(2)}$	0	1	1	1	1	2	2
$S_i^{(3)}$	0	0	1	1	2	2	2

After processing six stages of the trellis ($i = 6$), the state with the smallest metric is $S_6^{(0)}$ with associated (survivor) path $\bar{y}_6^{(0)} = \bar{v}$. One error has been corrected.

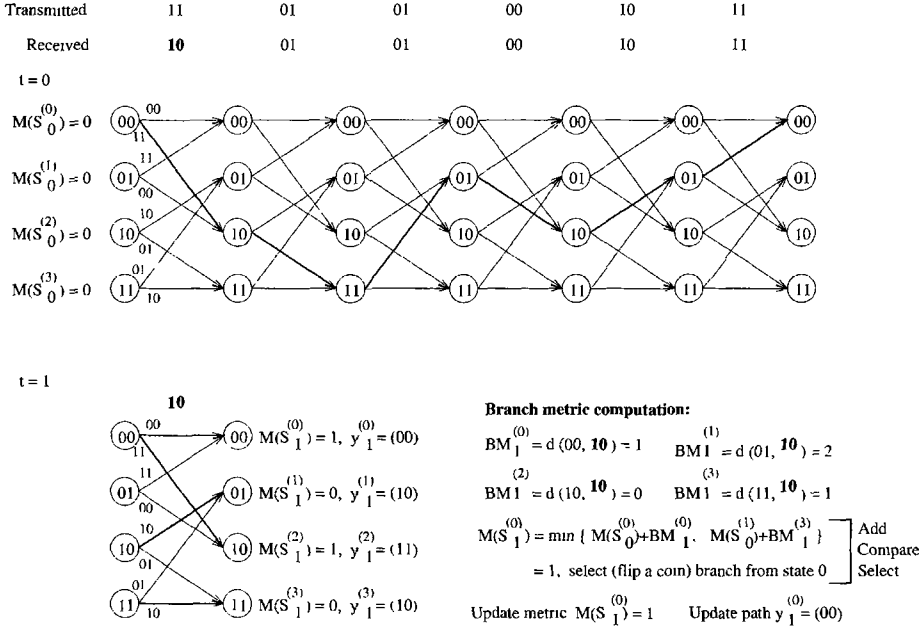


Figure 40 VD operation for Example 62, at $i = 0$ and $i = 1$.

5.4.3 Implementation issues

In this section, some of the implementation issues related to Viterbi decoders are discussed. The techniques below apply equally to any Viterbi decoder that operates over channels with additive metrics, such as the BSC, AWGN and flat Rayleigh fading channels.

Path metric initialization

The Viterbi decoder can operate *in the same mode* from the start ($i = 0$). The survivor paths can have arbitrary values, without affecting the decoder's performance. The first L decoded bits are therefore random and give no information. For this reason, the value of L contributes to the decoding delay, and is also known as the *decoding depth*. Moreover, provided that L is large enough ($L \gg \ell$, where $\ell > 5m$ for rate-$1/2$ binary codes), the decoded bit can be

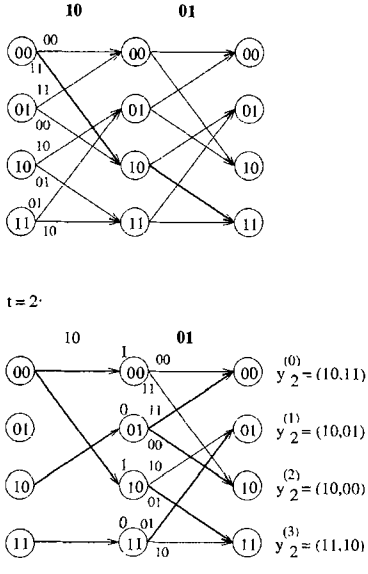


Figure 41 VD operation for Example 62, at $i = 2$.

output either from the path with the lowest metric or always output from the zero state path ($\bar{y}^{(0)}$). The latter method is easier to implement and does not result in loss of performance. The programs on the ECC web site that implement MLD using the Viterbi algorithm work in this fashion.

Also note that in Example 62 the branch labels (output) were stored in the survivor paths. This was done in order to facilitate understanding of the algorithm. In a practical implementation, however, it is the corresponding *information bits* that are stored. This is discussed below, in connection with path memory management.

Synchronization

Branch symbols must be properly *aligned* with the received symbols. Any misalignment can be detected by monitoring the value of a random variable associated with the Viterbi decoder. Two commonly used *synchronization variables* are: (1) Path metric growth, and (2) channel BER estimates. The statistics of these variables give an indication of abnormal decoding behavior.

Assume that the received sequence is not properly received. i.e., the n-bit branch labels $\bar{v}[i]$ in the decoder are not properly aligned, or synchronized, with the received sequence $\bar{r}[i]$.

Example 63 Figure 46 shows an example for a rate-$1/2$ in which the received sequence $\bar{r}$ is not synchronized with the reference coded sequence $\bar{v}$.

In other words, not all the bits in the received subsequence $\bar{r}[i]$ belong to the same trellis stage in the decoder. In this case, two events that may occur are: (1) The path metrics are close to each other and grow rapidly, and (2) the estimated channel BER approaches $1/2$. Figure 47 shows the block diagram of a Viterbi decoder and a BER monitor.

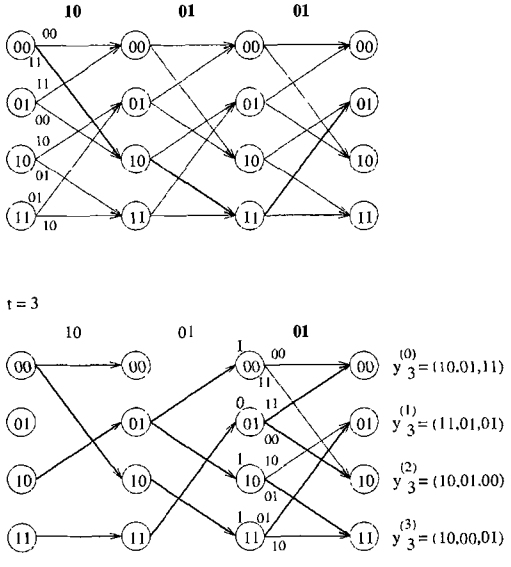


Figure 42 VD operation for Example 62, at $i = 3$.

A synchronization stage needs to be added, external to the decoder itself, whose function is to advance the reference sequence $\bar{v}$ in the decoder until the statistics return to normal. This can be done by skipping received symbols (a maximum of $n - 1$ times) until the synchronization variables indicate normal decoding behavior. This is indicated in Figure 46 of Example 63 for the case of a rate-$1/2$ convolutional code.

Metric normalization

As the Viterbi decoder operates continuously, the path metrics will grow proportionally with the length on the received sequence. To avoid overflow or saturation (depending on the number representation used), the metrics need to be normalized. There are basically two methods of doing this. Both rely on the following two properties of the Viterbi algorithm:

1. The MLD path selection depends only on the *metric differences*.
2. The metric differences are bounded.

- **Threshold method**
  To normalize the metrics, at each decoding stage, the value of the smallest metric

$$M_{\min} = \min_{0 \le k \le 2^m - 1} \left\{ M(S_i^{(k)}) \right\}$$

  is compared to a threshold T.If $M_{\min} > T$ then T is subtracted from all the metrics. Clearly, this does not affect the selection process, because the metric differences remain the same. This is a method that can be easily implemented in software on a generic processor.
- **Modular arithmetic method**
  The metrics are computed modulo N, so that they lie within the range $[0, N - 1]$, where $N = 2\Delta_{\max}$, and $\Delta_{\max}$ is the maximum difference in survivor path metrics.

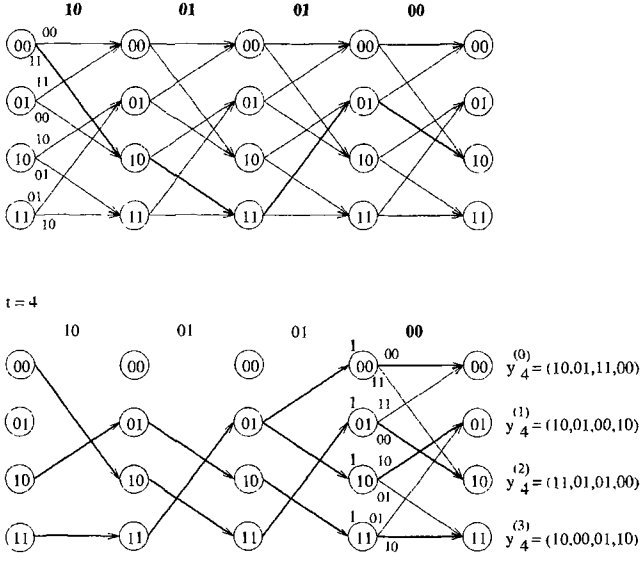


Figure 43 VD operation for Example 62, at $i = 4$.

Obviously, $\Delta_{\max}$ depends on the range of values received from the channel. From the two properties of the Viterbi algorithm, it can be shown that the same MLD path is selected by the Viterbi decoder when computing path metrics with modular arithmetic. In particular, it is possible to use *two's complement arithmetic*[7], in such a way that overflow can occur but it does not affect the selection process. For details, see [HEK, OCC]. This method is favored in hardware implementations of Viterbi decoders.

Path memory management

In a Viterbi decoder, survivor paths and their metrics need to be stored and updated. In a continuous operation mode, while updating the survivor paths at stage i, earlier portions of the survivor paths merge with high probability at stage $i - L$, where L is the decoding depth. The estimated information bits are taken from the (single) portion of the merged paths at stage $i - L$. There are different techniques to extract the information bits. Two of the most common are: (1) Register exchange and (2) Traceback memory.

- *Register exchange*
  This method is the easiest to implement in software. All the survivor paths are updated at each iteration of the Viterbi algorithm. Therefore, the information bits can be read directly from the survivor paths. However, if this technique is implemented in hardware, the decoding speed would be low, due to an excessive number of times at which the path memory is read and written. To simplify control flow instructions, a *circular pointer* of length L can be used. With a circular pointer, at decoding stage i, position $(i - L)$ in memory (to output the decoded bit) is equal to position $(i + 1)$

[7] Arithmetic using the additive group $\{-2^{m-1}, 1 - 2^{m-1}, \cdots, 2^{m-1} - 1\}$ of m-bit integers [HEK].

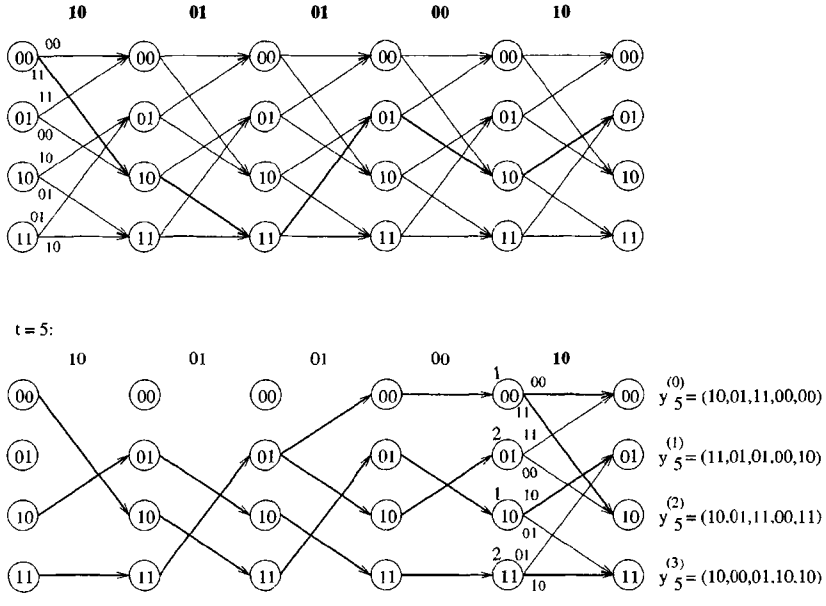


Figure 44 VD operation for Example 62, at $i = 5$.

modulo L.

- *Traceback*

  This technique is favored in implementations in hardware. The survivor paths are composed of *decision values*, which indicate *state transitions* in order to trace back the survivor paths and reconstruct a *sequences of states* in reverse order.

  The traceback memory can be organized as a rectangular array, with rows indexed by k, for all the trellis states $S_i^{(k)}$, $k = 0, 1, \cdots, 2^m - 1$. At decoding stage i, for each trellis state $S_i^{(j)}$, $0 \leq j \leq 2^m - 1$, the *traceback memory* (or TB RAM) is written with the rightmost bit of the previous state $S_{i-1}^{(b_j)}$, $\ell \in \{1, 2\}$, associated with the winning branch.

  The traceback method trades off memory for decoding speed, since it writes only one bit per state per decoding stage, instead of L bits per state per decoding as in the register exchange method. The information bits are decoded by reading in reverse order the state transitions with an encoder replica (or looking up a state transition table). More memory (number of columns if organized in a rectangular array) is required for continuous operation because, at the same time as information bits are being read out (decoded), new decision values are being written.

Example 64 Continuing with Example 62, using the traceback technique, at the end of the $i = 6$ received pair of bits from the channel, the traceback memory would have the contents shown in Table 3.

The traceback memory is read starting from the last bit (e.g., use a LIFO), $i = 6$ (in general $T = L$). The row address is given by the state with the best metric. This is state $S^{(0)}$ in the example. Read the *transition bit* b_6 (in general, b_L). The row address (ROW) at $i = 5$ ($i = L - 1$), to read the next transition bit b_5, is obtained by shifting the address k at $i = 6$

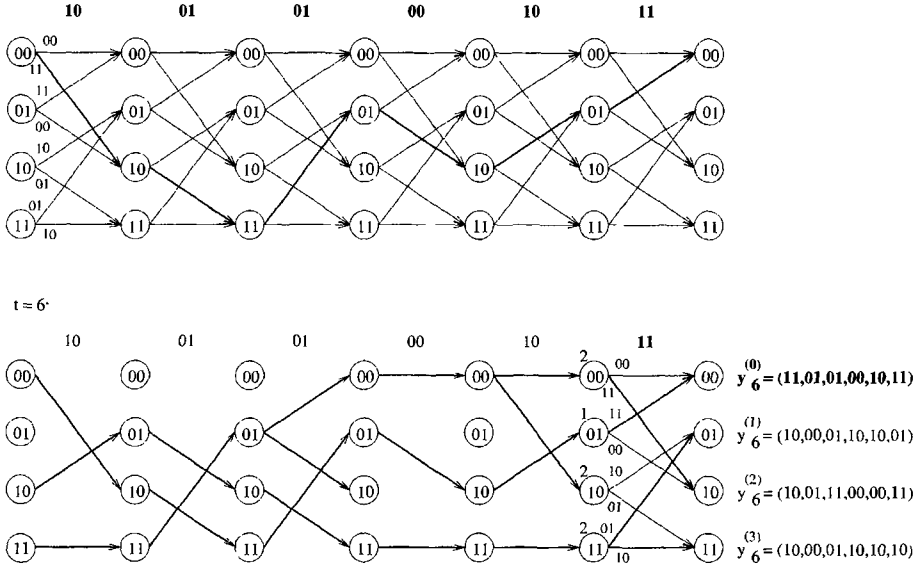


Figure 45 VD operation for Example 62, at $i = 6$.

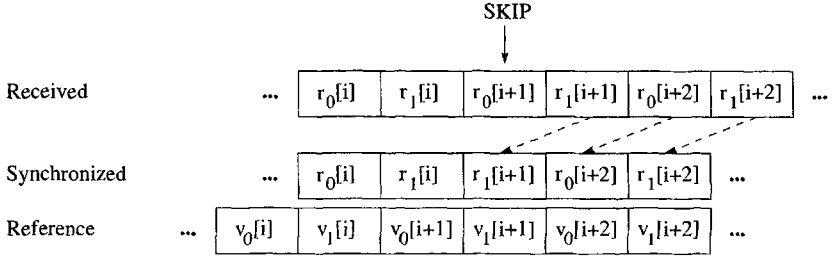


Figure 46 Example of branch misalignment in a Viterbi decoder.

($i = L$) and appending the previous decision bit b_6 (b_L). This can be stated in the following C language instruction:

```
ROW[j] = ((ROW[j+1] << 1) && MASK) ^ TB_RAM[ROW[j+1]][j+1];
```

(where MASK $= 2^m - 1$.)

Continuing in this way, the following state sequence is obtained:

$$S_6^{(0)} \rightarrow S_5^{(1)} \rightarrow S_4^{(2)} \rightarrow S_3^{(1)} \rightarrow S_2^{(3)} \rightarrow S_1^{(2)} \rightarrow S_0^{(0)}.$$

From this sequence, the corresponding information sequence can be recovered, by reversing the state transitions: $\hat{u} = (1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0)$.

For high-speed continuous VD operation with the traceback method, the traceback memory must be partitioned into several blocks. In this way, while one block is being written with decisions at the current stage i, another block is read (decoded) at a time $i - L$, $L > \ell$, and another (possibly more than one) block for intermediate stages is used for performing

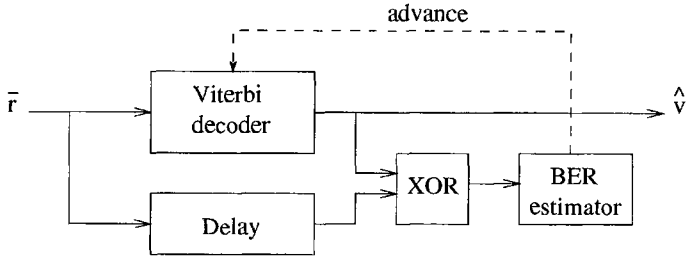


Figure 47 Channel error rate estimation for a BSC channel.

Table 3 Tracing back the path that ended in state $S^{(0)}$.

k	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
0	0	1	1	0	0	**1**
1	0	1	**1**	0	**0**	1
2	**0**	1	1	**1**	0	0
3	1	**0**	0	1	1	1

traceback tracing. This memory partitioning improves speed but increases the latency (or decoding delay) of the decoder [Col, BABSZ].

ACS

For memory-m rate-$1/n$ binary convolutional codes codes, the basic trellis element is a *butterfly*, as shown in Figure 48. The add-compare-select (ACS) operation is implemented using this structure over 2^{m-1} pairs, i.e., $j = 0, 1, 2, \cdots, 2^{m-1} - 1$. Therefore, the ACS operation can be done either serially (a loop on j in software) or in parallel, with 2^{m-1} ACS units, one per butterfly. If the code is *antipodal* then the generators of the code have a one in the first and last positions. In this case, the labels of the branches incident to a state $S^{(2j)}$ are the same as those incident to state $S^{(2j-1)}$ in a butterfly. Moreover, the label of a branch incident to a state $S^{(2j)}$ is equal to the complement of the label of the other branch[8]. Using these facts, a clever technique [FL2] based on branch metric differences was proposed. For rate-$1/n$ codes, the result is a Viterbi decoder with a *compare-add-select* (CAS) architecture, and lower complexity of implementation than the conventional ACS approach.

5.5 Punctured convolutional codes

Punctured convolutional codes were introduced in [Cla]. Puncturing is the process of systematically deleting, or not sending, some output bits of a low-rate encoder[9]. Since the trellis structure of the low-rate encoder remains the same, the number of information bits per sequence does not change. As a result, the output sequences belong to a higher-rate *punctured convolutional (PC) code*. The following discussion focuses on codes obtained from a binary

[8] The complement of a bit a is $1 + a \bmod 2$.
[9] Sometimes referred to as the *mother code*.

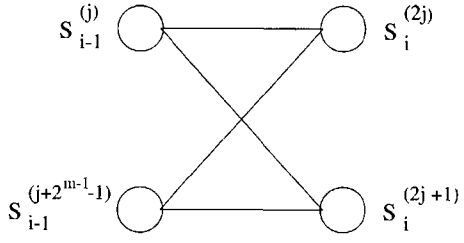


Figure 48 Trellis butterfly of a memory-m rate-$1/n$ binary convolutional code.

memory-m rate-$1/n$ convolutional code.

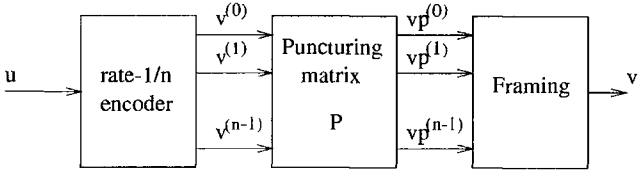


Figure 49 Encoder of a punctured code based on a rate-$1/n$ code.

A *puncturing matrix* P specifies the rules of deletion of output bits. P is a $k \times n_p$ binary matrix, with binary symbols p_{ij} that indicate whether the corresponding output bit is transmitted ($p_{ij} = 1$) or not ($p_{ij} = 0$). Generally, the puncturing rule is periodic. A rate-k/n_p PC encoder based on a rate-$1/n$ encoder, has a puncturing matrix P that contains ℓ zero entries, where $n_p = kn - \ell$, $0 \leq \ell < kn$.

Example 65 A rate-2/3 memory-2 convolutional code can be constructed by puncturing the output bits of the encoder of Figure 29, according to the puncturing matrix

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

The corresponding encoder is depicted in Figure 50. A coded sequence

$$\bar{v} = (\cdots, v_i^{(0)} v_i^{(1)}, v_{i+1}^{(0)} v_{i+1}^{(1)}, v_{i+2}^{(0)} v_{i+2}^{(1)}, v_{i+3}^{(0)} v_{i+3}^{(1)}, \cdots)$$

of the rate-$1/2$ encoder is transformed into code sequence

$$\bar{v}_p = (\cdots, v_i^{(0)} v_i^{(1)}, v_{i+1}^{(0)}, v_{i+2}^{(0)} v_{i+2}^{(1)}, v_{i+3}^{(0)}, \cdots),$$

i.e., every other bit of the second output is not transmitted.

One of the goals of puncturing is that the same decoder can be used for a variety of high-rate codes. One way to achieve decoding of a PC code using the Viterbi decoder of the low-rate code, is by the insertion of "deleted" symbols in the positions that were not sent. This process is known as *depuncturing*. The "deleted" symbols are marked by a special flag. This can be done, for example, using an additional bit and setting it to 1 in the position of an erased bit. If

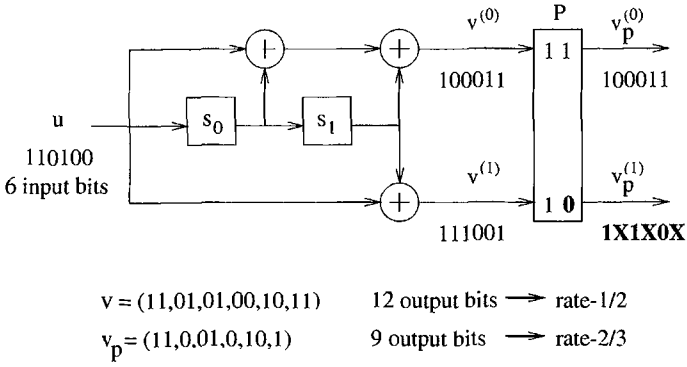


Figure 50 Encoder of a rate-2/3 memory-2 PCC.

Table 4 Puncturing matrices for the de-facto standard rate-1/2 code.

Rate	Pattern	Coded sequence	d_f
1/2	$\begin{pmatrix} 1 \\ 1 \end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}$	10
2/3	$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}, v_{i+1}^{(1)}$	6
3/4	$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}, v_{i+1}^{(1)}, v_{i+2}^{(0)}$	5
5/6	$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}, v_{i+1}^{(1)}, v_{i+2}^{(0)}, v_{i-3}^{(1)}, v_{i+4}^{(0)}$	4
7/8	$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}, v_{i-1}^{(1)}, v_{i+2}^{(1)}, v_{i+3}^{(1)}, v_{i+4}^{(0)}, v_{i+5}^{(1)}, v_{i-6}^{(0)}$	3

a position is flagged, then the corresponding received symbol is not taken into account in the branch metric computation[10].

In a software implementation, an alternative method is for the decoder to check the entries $p_{m,j}$ of matrix P in the transmission order, $m = 0, 1, \cdots, n-1$, $j = i, i+1, \cdots, i+k-1$. If $p_{m,j} = 0$, then the current received symbol is not used to update the branch metric. The decoder advances a pointer to the next branch symbol in the trellis and repeats the test on $p_{m,j}$. Otherwise, $p_{m,j} = 1$ and the received symbol are used to update the branch metric. Every time the branch pointer is advanced, a check is made to determine if all the symbols in that branch have been processed. If they have, then the ACS operation is performed. This method is used in some of the programs that implement Viterbi decoding of PC codes on the ECC web site.

Table 4 shows puncturing matrices employed with the de-facto standard memory-6 rate-1/2 convolutional code with generators $(\bar{g}_0, \bar{g}_1) = (171, 133)$. In the table, $v_m^{(i)}$ indicates the output, at time m, associated with generator $\bar{g}_i$, $i = 0, 1$. Other puncturing matrices can be found in [YKH].

[10] Some authors write "the branch metric is zero" as an equivalent statement.

Example 66 (Continuation of Example 65.) Let $\bar{u}$ be the same as in Example 62, and suppose transmission over a BSC introduces an error in the second position. Coded symbols $v^{(1)}[i], i = 0, 2, 4, \cdots$, are not transmitted. The decoder knows this and inserts deleted symbols, denoted by E. When computing branch distances, the E symbols are ignored. The operation of a Viterbi decoder with deleted symbols is depicted in Figures 51 and 52 for $i = 1$ and $i = 2$, respectively. It is left as an exercise to finish the example.

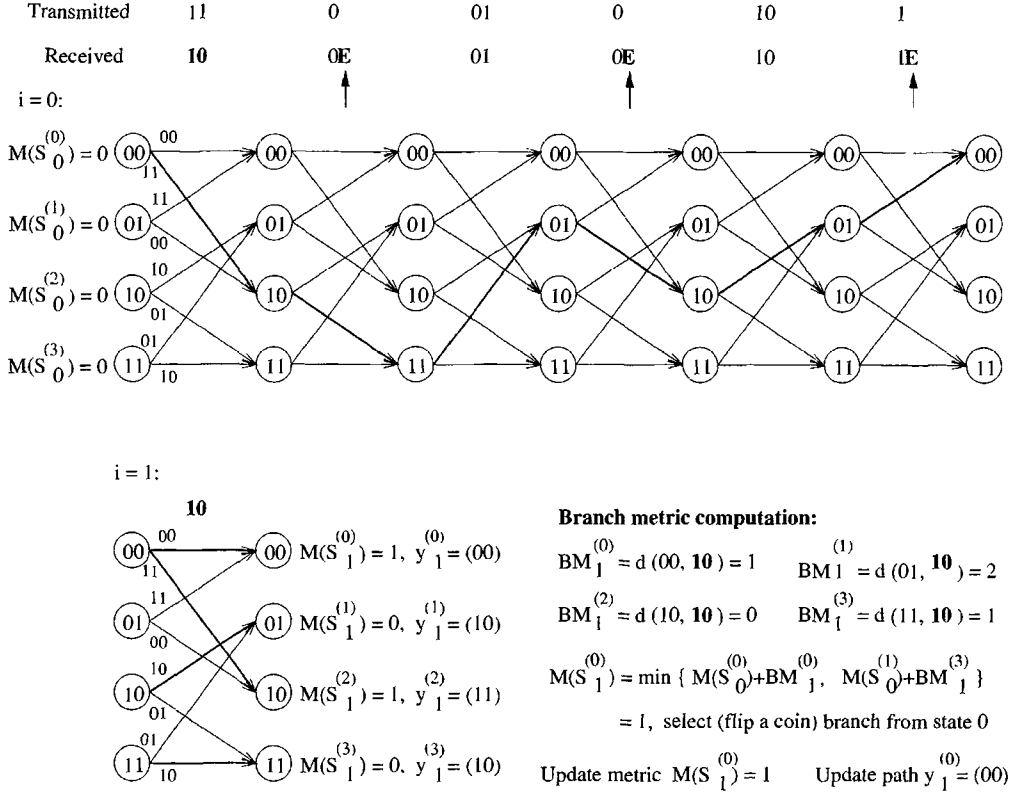

Transmitted 11 0 01 0 10 1
Received 10 0E 01 0E 10 1E
i = 0:
i = 1:
10
Branch metric computation:
BM(0)1 = d (00, 10) = 1
BM(1)1 = d (01, 10) = 2
BM(2)1 = d (10, 10) = 0
BM(3)1 = d (11, 10) = 1
M(S(0)1) = min { M(S(0)0)+BM(0)1, M(S(1)0)+BM(3)1 }
= 1, select (flip a coin) branch from state 0
Update metric M(S(0)1) = 1
Update path y(0)1 = (00)

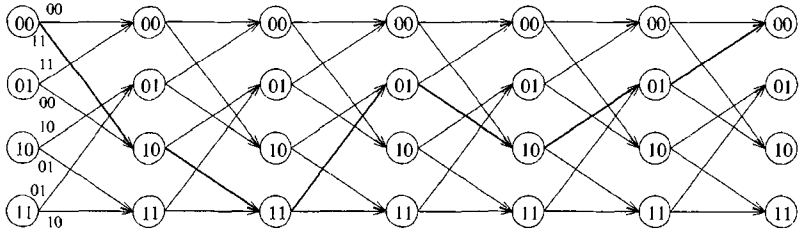
Figure 51 VD operation for a rate-2/3 memory-2 PCC, $i = 1$.

5.5.1 Implementation issues related to punctured convolutional codes

For a system that employs punctured convolutional codes, the re-synchronization process will take longer, as the output symbols are spread over several trellis branches. Moreover, practical systems use a multiplicity of puncturing patterns of different lengths. The decoder therefore may have to make a guess as to the specific rate and puncturing pattern being received. Other points to consider when implementing PC codes are:

1. The *decoding depth L must be increased* as more output bits are punctured. This is because high-rate codes (e.g., rate 7/8) have low minimum Hamming distance

Transmitted	11	0	01	0	10	1
Received	**10**	**0E**	01	0E	10	1E



i = 2:

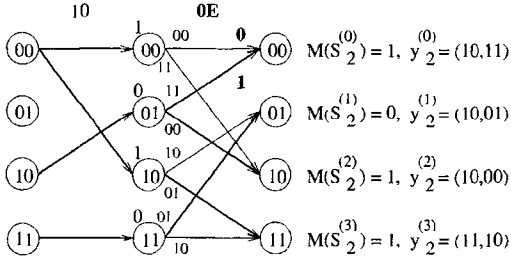


Figure 52 VD operation for a rate-2/3 memory-2 PCC, $i = 2$.

between coded sequences (e.g., $d_{min} = 3$) and therefore the survivor paths take longer to converge.

2. An *additional level of synchronization* with the received sequence is needed, to align the puncturing pattern.

The fact that the puncturing patterns have different lengths for different rates may be solved by employing the same *puncturing period*. This is one of the features of *rate-compatible* PC codes, or RCPC codes.

5.5.2 *RCPC codes*

RCPC codes were introduced by Hagenauer [Hag]. These codes are constructed from a low-rate code by periodic puncturing. Let M denote the puncturing period. Then, as before, the puncturing matrices are $n \times M$ binary matrices. However, in general, $M > kn - \ell$.

To construct a family of RCPC codes, it is required that the branch outputs of a high-rate PC code are used by the lower rates PC codes. This is achieved as follows. Let $P^{(H)}$ and $P^{(L)}$ denote matrices of a high-rate code C_H and a low-rate code C_L, respectively, both derived from the same lowest-rate code C. Then codes C_H and C_L are *rate-compatible* if the following condition holds:

$$p_{ij}^{(H)} = 1 \rightarrow p_{ij}^{(L)} = 1.$$

An equivalent condition is

$$p_{ij}^{(L)} = 0 \rightarrow p_{ij}^{(H)} = 0.$$

Example 67 Let C be a rate-$1/2$ convolutional code. Let $M = 4$. Then matrices $P(1)$ through $P(4)$ below generate RCPC codes of rates 4/5, 4/6, 4/7 and 4/8(=1/2), respectively.

$$P(1) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \quad P(2) = \begin{pmatrix} 1 & 1 & \mathbf{1} & 0 \\ 1 & \mathbf{1} & 0 & 1 \end{pmatrix},$$
$$P(3) = \begin{pmatrix} 1 & 1 & 1 & \mathbf{1} \\ 1 & 1 & 0 & 1 \end{pmatrix}, \quad P(4) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & \mathbf{1} & 1 \end{pmatrix}.$$

RCPC codes [Hag] find applications in automatic repeat request (ARQ) systems, due to their incremental redundancy nature, as well as in constructing variable-rate or *unequal error protection* codes.

6

Modifying and combining codes

In this chapter, several techniques are presented that allow modification of an existing code, or the combination of several codes, in order to achieve flexibility in the design of error correcting codes. Many of the best codes known to date have been obtained not from members of a known family of codes, but from modifying and combining codes [Bro].

6.1 Modifying codes

In the following, let C denote a linear block (n, k, d) code over $GF(q)$ with generator matrix G and parity-check matrix H.

6.1.1 Shortening

Shortened cyclic codes were described in Section 3.1.5. Let s be an integer, $0 \leq s < k$. In general, a linear shortened $(n - s, k - s, d_s)$ code C_s has distance $d_s \geq d$. A generator matrix of C_s can be obtained from the generator matrix G of the original code C as follows. Assume that G is in systematic form, that is,

$$G = (I_k | P) . \tag{6.1}$$

Then a $(k - s) \times (n - s)$ generator matrix G_s of the shortened code C_s is obtained by removing s columns of the identity matrix I_k and the s rows that correspond to where the selected columns elements are nonzero. This is best illustrated by an example.

Example 68 Consider the Hamming $(7, 4, 3)$ code, presented in Example 13, page 23, of Chapter 2. This code has

$$G = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & 1 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & 0 & 1 & 0 & 1 & 1 \end{pmatrix} . \tag{6.2}$$

(Compare with the H matrix in Example 13.)

To construct a shortened $(5, 2, 3)$ code, any two among the four leftmost columns of G can be removed. Suppose that the first and second columns of G are removed, and that the first and second rows, corresponding to the nonzero elements of the columns, are removed. These rows and columns are indicated by boldface types in (6.2). The remaining entries of G form

the matrix

$$G_s = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

It is instructive to examine the standard array of the shortened (5,2,3) code of Example 68, to understand the enhancement in the error correcting capability of a shortened code, with respect to the original code.

Example 69 The standard array for the shortened $(5, 2, 3)$ code of Example 68 is given in Table 5.

Table 5 Standard array for a shortened $(5, 2, 3)$ code.

$\bar{s}$	$\bar{u} = 00$	$\bar{u} = 10$	$\bar{u} = 01$	$\bar{u} = 11$
000	00000	10110	01011	11101
110	10000	00110	11011	01101
011	01000	11110	00011	10101
100	00100	10010	01111	11001
010	00010	10100	01001	11111
001	00001	10111	01010	11100
101	11000	01110	10011	00101
111	01100	11010	00111	10001

From the standard array, it follows that, although the minimum distance of the code is $d_s = 3$, there are two error patterns of Hamming weight two, namely 11000 and 01100, that can be corrected.

Note that shortening a code reduces the length and dimension of the code, while at the same time maintains the same redundancy, so that more error patterns can be corrected. This can be explained from the Hamming bound for a t-error-correcting linear block (n, k, d) code, inequality (1.24) which is repeated here for convenience,

$$2^{n-k} \geq \sum_{\ell=0}^{t} \binom{n}{\ell}. \tag{6.3}$$

With respect to the original (n, k, d) code, the shortened $(n - s, k - s, d_s)$ has the same redundancy. Therefore, the left-hand side of (6.3) has the same value. In other words, *the number of cosets does not change.* On the other hand, for $s > 0$, the right-hand side becomes smaller. In other words, *the number of error patterns of Hamming weight up to t decreases.*

If the original code does not meet the Hamming bound (6.3) with equality (in the binary case, only the Hamming codes, the Golay code, the single parity-check codes and the repetition codes do), then the quantity $2^{n-k} - \sum_{i=0}^{t} \binom{n}{i}$ represents the additional patterns of weight greater than t that the original code can correct. The number of additional error patterns of Hamming weight greater than t that the shortened code can correct is given by

$$\Delta_t = 2^{n-k} - \sum_{\ell=0}^{t} \binom{n-s}{\ell} - \left[2^{n-k} - \sum_{\ell=0}^{t} \binom{n}{\ell} \right] = \sum_{\ell-0}^{t} \left[\binom{n}{\ell} - \binom{n-s}{\ell} \right], \tag{6.4}$$

which is equal to the difference in the volume of Hamming spheres of radius t when going from n to $n - s$ coordinates (or dimensions).

6.1.2 *Extending*

In general, extending a code C means adding ϵ parity-check symbols. The extended $(n+\epsilon, k, d_{\text{ext}})$ code C_{ext} has minimum distance $d_{\text{ext}} \geq d$. In terms of the parity-check matrix H of the code C, ϵ rows and columns are added to it, to obtain an $(n-k+\epsilon) \times (n+\epsilon)$ extended parity-check matrix,

$$H_{\text{ext}} = \begin{pmatrix} h_{1,1} & \cdots & h_{1,\epsilon} & \cdots & h_{1,n+\epsilon} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{\epsilon,1} & \cdots & h_{\epsilon,\epsilon} & \cdots & h_{\epsilon,n+\epsilon} \\ \vdots & \ddots & \vdots & & H \\ h_{n-k+\epsilon,1} & \cdots & h_{n-k+\epsilon,\epsilon} & & \end{pmatrix}. \tag{6.5}$$

The most common way of extending a code is by adding an overall parity-check symbol. In this case, the following parity-check matrix is obtained,

$$H_{\text{ext}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & & & \\ 0 & & H & \\ 0 & & & \end{pmatrix}. \tag{6.6}$$

The resulting code C_{ext} is an $(n+1, k, d_{\text{ext}})$ code. If the minimum distance of the original code is odd, then $d_{\text{ext}} = d+1$.

Example 70 Let C be the Hamming (7,4,3) code. Then the extended (8,4,4) code C_{ext} has parity-check matrix

$$H_{\text{ext}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{6.7}$$

After permutation of columns, this is also the generator matrix of the $\text{RM}_{1,3}$ code of Example 16, which is a *self-dual code*[1].

6.1.3 *Puncturing*

A puncturing technique was discussed in Section 5.5, in connection with convolutional codes. More generally, puncturing of linear block codes consists of the removal of parity check symbols, to obtain a linear block $(n-p, k, d_p)$ code C_p with minimum distance $d_p \leq d$. The rate of the code increases, since the dimension does not change, and redundancy (number of parity-check symbols) is reduced.

Puncturing is achieved by removing certain columns of the parity-check matrix H of the original code, to obtain matrix H_p of C_p. This technique is similar to shortening of the dual code. If

$$H = (\bar{p}_0 \quad \bar{p}_2 \quad \cdots \quad \bar{p}_{k-1} | I_{n-k}),$$

[1] See page 7.

denotes the parity-check matrix of C, then removing any p columns among the $n-k$ rightmost columns from H, and p rows corresponding to the nonzero values of the selected columns, results in an $(n-k-p) \times (n-p)$ matrix

$$H_p = (\vec{p}'_0 \quad \vec{p}'_1 \quad \cdots \quad \vec{p}'_{k-1} | \bar{I}_{j_0} \quad \bar{I}_{j_1} \quad \cdots \quad \bar{I}_{j_{k-p-1}}),$$

where $\bar{I}_j$ denotes a column of weight equal to one, and $\vec{p}'_j$ denotes a column after the removal of the p rows described above.

Example 71 Consider the $(5, 2, 3)$ code C_s from Example 68. Its parity-check matrix is equal to

$$H_s = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & \mathbf{0} & 1 & 0 \\ 0 & 1 & \mathbf{0} & 0 & 1 \end{pmatrix}.$$

Deleting the third column and the top row of H_s gives the parity-check matrix H_p of a punctured $(4, 2, 2)$ code C_p.

$$H_p = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix},$$

which is the same LUEP code as in Examples 3 and 6, up to a permutation of the first and second code positions.

6.1.4 Augmenting and expurgating

There are two additional techniques to modify an existing linear code that in general may produce nonlinear codes. These techniques are *augmenting* and *expurgating* and are very useful in analyzing families of linear block codes, such as the Reed-Muller and BCH codes.

Augmenting a code means adding more codewords to it. An obvious way to do this and still get a linear code is to add more (linearly independent) rows to the generator matrix. This is equivalent to forming a *supercode* by taking the union of *cosets* of code C. The resulting code C_{aug} has dimension $(n, k+\delta, d_{\text{aug}})$, where δ is the number of rows added to G. The minimum distance of C_{aug} is $d_{\text{aug}} \leq d$. Let $G = (\bar{g}_0^\top \quad \cdots \bar{g}_{k-1}^\top)^\top$ be a generator matrix of C, where $\bar{g}_j$ denotes a vector of length n, $0 \leq j < n$, and $\bar{x}^\top$ denotes the *transpose* of a vector $\bar{x}$. Then a generator matrix of an augmented code is given by

$$G_{\text{aug}} = \begin{pmatrix} \bar{p}_0 \\ \vdots \\ \bar{p}_{\delta-1} \\ \\ G \end{pmatrix} \tag{6.8}$$

Augmenting techniques are closely related to *coset decomposition* [For6] of codes. This is discussed below in connection with the direct-sum of codes and the $|u|u+v|$-construction and related techniques. A common method to augment a linear block (n, k, d) code C is to add the all-one codeword to it (if it is not already in the code.) Alternatively, if the all-one codeword belongs to C, then

$$C = C_0 \bigcup \{\bar{1} + C_0\},$$

for an $(n, k-1, d_0)$ subcode $C_0 \subset C$, where $d_0 = d-1$.

Table 6 Basic techniques for modifying a code.

Technique	Action	Code parameters
Shortening	Removing information symbols	$(n-\ell, k-\ell, d_s \geq d)$
Lengthening	Adding information/parity-check symbols	$(n+\ell, k+\ell, d_s \leq d)$
Extending	Adding parity-check symbols	$(n+\ell, k, d_{\text{ext}} \geq d)$
Puncturing	Removing parity-check symbols	$(n-\ell, k, d_p \leq d)$
Augmenting	Adding codewords (cosets)	$(n, k+\ell, d_{\text{aug}} \leq d)$
Expurgating	Removing codewords (cosets)	$(n, k-\ell, d_{\text{exp}} \geq d)$

Expurgating is the process of removing codewords from a code. In general, this will produce a nonlinear code. One way to get a linear code is to remove rows from the generator matrix G of the original code. This results in an $(n, k-\ell, d_{\text{exp}})$ code C_{exp} with $d_{\text{exp}} \geq d$. It is interesting to note that if the code is systematic, then expurgating is equivalent to shortening.

Example 72 Consider the Hamming $(7,4,3)$ code, with

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}. \tag{6.9}$$

Removing the first row results in a $(7,3,3)$ code, for which the first bit is always zero. In other words, the code is equivalent to a $(6,3,3)$ code.

It is important to note that when expurgating a code, the choice of the generator matrix, or the *basis of the original code*, is important in order to obtain codes with high minimum distance. This is illustrated in the following example.

Example 73 As shown in Example 39, the Hamming $(7,4,3)$ code contains seven codewords of weight 3 and seven codewords of weight 4. Selecting four linearly independent codewords, three of weight 4 and one of weight 3, the following generator matrix is obtained,

$$G' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{6.10}$$

Matrix G' generates the same Hamming $(7,4,3)$ of the previous example. However, if the top row of G' is removed, the generator matrix of a maximum-length-sequence (or *simplex*) $(7,3,4)$ code is obtained.

The code *lengthening* technique is performed by adding ℓ columns and ℓ rows to the generator matrix, and ℓ additional parity-check symbols, so that ℓ additional information symbols are introduced. In particular, one way to lengthen a code is to add one information symbol and one overall parity-check symbol [MS].

Table 6 summarizes the techniques presented in this section.

6.2 Combining codes

In this section, several methods of combining codes are presented. Code combining techniques are very powerful, as evidenced by the appearance in 1993 of *turbo codes* [BGT]. In the following, unless otherwise specified, let C_i denote a linear block (n_i, k_i, d_i) code, $i = 1, 2$.

6.2.1 *Time-sharing of codes*

Consider two codes C_1 and C_2. Then the *time-sharing* of C_1 and C_2 consists of transmitting a codeword $\bar{c}_1 \in C_1$ followed by a codeword $\bar{c}_2 \in C_2$,

$$|C_1|C_2| = \{(\bar{c}_1, \bar{c}_2) : \bar{c}_i \in C_i,\ i = 1, 2\}. \tag{6.11}$$

The result of time-sharing of m linear block (n_i, k_i, d_i) codes, $i = 1, 2, \cdots, m$, is an (n, k, d) code, with parameters

$$n = \sum_{i=1}^{m} n_i, \quad k = \sum_{i=1}^{m} k_i, \text{ and } d = \min_{1 \le i \le m} \{d_i\}. \tag{6.12}$$

Let G_i denote the generator matrix of *component code* C_i, for $i = 1, 2, \cdots, m$. Then the generator matrix of the code obtained from time-sharing is

$$G_{\mathrm{TS}} = \begin{pmatrix} G_1 & & & \\ & G_2 & & \\ & & \ddots & \\ & & & G_m \end{pmatrix}, \tag{6.13}$$

where the blank entries represent zeros.

Time-sharing is sometimes referred to as "direct-sum" [MS] or "concatenation" [Rob]. In this book, however, concatenation of codes has a different interpretation, and is discussed in Section 6.2.4.

Example 74 Let C_1 be a repetition $(4, 1, 4)$ code and C_2 be a Hamming $(7, 4, 3)$ code. Then time-sharing of C_1 and C_2 results in a linear block $(11, 5, 3)$ code with generator matrix

$$G_{\mathrm{TS}} = \begin{pmatrix} G_1 & 0 \\ 0 & G_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

The time-sharing technique has been widely used in communication systems that require a variable amount of error protection, or *unequal error protection*, using the RCPC codes of Section 5.5.2. See [Hag]. Also note that time sharing m times the same code is equivalent to repeating the transmission of a codeword m times. More on this when product codes are discussed, in Section 6.2.3.

6.2.2 *Direct-sums of codes*

Let C_i denote a linear block (n, k_i, d_i) code, $1 \leq i \leq m$. A direct-sum code C_{DS} is defined as

$$C_{\mathrm{DS}} = \{\bar{v} | \bar{v} = \bar{v}_1 + \bar{v}_1 + \cdots + \bar{v}_m, \ \ \bar{v}_i \in C_i, \ \ i = 1, 2, \cdots, m\}.$$

This technique may increase dimension. However, it generally reduces the distance. Let G_i denote the generator matrix of component code C_i, for $i = 1, 2, \cdots, m$. Then the generator matrix of the code C_{DS}, obtained from the direct-sum of these component codes, denoted $C_{\mathrm{DS}} = C_1 + C_2 + \cdots + C_m$, is

$$G_{\mathrm{DS}} = \begin{pmatrix} G_1 \\ G_2 \\ \vdots \\ G_m \end{pmatrix}. \tag{6.14}$$

Code C_{DS} is a linear block (n, k, d) code with $k \leq k_1 + k_2 + \cdots + k_m$ and $d \leq \min_i\{d_i\}$.

Example 75 Let C_1 be the repetition $(4, 1, 4)$ code and C_2 be a linear block $(4, 2, 2)$ code with generator matrix

$$G_2 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

(This code consists of codewords formed by sending twice a 2-bit message.) Then code $C_{\mathrm{DS}} = C_1 + C_2$ is a single parity-check $(4, 3, 2)$ code, with generator matrix,

$$G_{\mathrm{DS}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

The direct-sum technique can be used not only to combine codes of smaller dimension, but also to *decompose* a code into a *union of subcodes* $C_i \subset C$, such that C can be expressed as the direct sum of the component subcodes. More on this is discussed in Section 6.2.4.

Trivially, every linear block (n, k, d) code C with generator matrix G can be decomposed into k linear block $(n, 1, d_i)$ subcodes C_i, $1 \leq i \leq k$. Each subcode has a generator matrix equal to a row $\bar{g}_i$, of G, $0 \leq i < k$. However, there are known construction techniques that allow decomposition of codes into subcodes of larger dimension and known minimum distances. This is the topic of the next section.

The $|u|u + v|$-construction and related techniques

Combining time-sharing and direct-sum gives interesting construction methods. The first one is the $|u|u + v|$-*construction* [Plo], defined as follows. Based on two codes, C_1 and C_2, of length $n = n_1 = n_2$, the $|u|u+v|$-construction gives a code[2] $C \triangleq |C_1|C_1+C_2|$ with generator matrix

$$G = \begin{pmatrix} G_1 & G_1 \\ 0 & G_2 \end{pmatrix}. \tag{6.15}$$

[2] If the lengths are not equal, say, $n_1 > n_2$, then append $n_2 - n_1$ zeros to the end of codewords of code C_2.

The parameters of C are $(2n, k_1 + k_2, d)$. That the minimum distance of C is $d = \min\{2d_1, d_2\}$ can be proven by noticing that for two binary vectors $\bar{x}$ and $\bar{y}$, $\mathrm{wt}(\bar{x} + \bar{y}) \leq \mathrm{wt}(\bar{x}) - \mathrm{wt}(\bar{y})$.

The $|u|u + v|$-construction allows to express RM codes in a convenient way:

Reed-Muller codes

$$\mathrm{RM}(r+1, m+1) = |\mathrm{RM}(r+1, m)|\mathrm{RM}(r+1, m) + \mathrm{RM}(r, m)|. \tag{6.16}$$

This interpretation of RM codes has been used to develop efficient soft-decision decoding methods taking advantage of the "recursive" structure of this class of codes (see [For6, SB] and references therein). In Section 6.2.4, this construction will be shown to be a particular case of a more powerful combining technique. The $|u|u + v|$-construction is also called the *squaring construction* in [For6].

Example 76 Let C_1 be RM$(1, 2)$, an SPC $(4,3,2)$ code, and C_2 be RM$(0, 2)$, the repetition $(4, 1, 4)$ code. Then $C = |C_1|C_1 + C_2|$ is RM$(1, 3)$, an extended Hamming $(8, 4, 4)$ code with generator matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Construction X

This is a generalization of the $|u|u + v|$-construction [Slo]. Let C_i denote a linear block (n_i, k_i, d_i) code, for $i = 1, 2, 3$. Assume that C_3 is a subcode of C_2, so that $n_3 = n_2$, $k_3 \leq k_2$ and $d_3 \geq d_2$. Assume also that the dimension of C_1 is $k_1 = k_2 - k_3$. Let $(\, G_2^\top \quad G_3^\top \,)^\top$ and G_3 be the generator matrices of code $C_2 \supset C_3$ and subcode C_3, respectively. Note that G_2 is a *set of coset representatives* of C_3 in C_2 [For6]. Then the code C_X with generator matrix

$$G_X = \begin{pmatrix} G_1 & G_2 \\ 0 & G_3 \end{pmatrix} \tag{6.17}$$

is a linear block $(n_1 + n_2, k_1 + k_2, d_X)$ code with $d_x = \min\{d_3, d_1 + d_2\}$.

Example 77 Let C_1 be a single-parity check (SPC) $(3, 2, 2)$ code, and C_2 be an SPC $(4, 3, 2)$ code whose subcode is C_3, a repetition $(4, 1, 4)$ code. Then

$$G_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}; \quad \begin{pmatrix} G_2 \\ G_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix};$$

and $G_3 = (\, 1 \quad 1 \quad 1 \quad 1 \,)$. Construction X results in code $C_X = |C_1|C_2 + C_3|$ with generator matrix

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

and is a maximum-length-sequence $(7, 3, 4)$ code. This code is equivalent to the code obtained from the Hamming $(7, 4, 3)$ code by expurgating one message symbol, as in Example 73.

Construction X3

Extending further the idea of using coset representatives of subcodes in a code, this method combines three codes, one of them with two levels of *coset decomposition* into subcodes, as follows [Slo]. Let C_3 be a linear block (n_1, k_3, d_3) code, where $k_3 = k_2 + a_{23} = k_1 + a_{12} + a_{23}$. C_3 is the union of $2^{a_{23}}$ disjoint cosets of a linear block (n_1, k_2, d_2) code, C_2, with $k_2 = k_1 + a_{12}$. In turn, C_2 is the union of $2^{a_{12}}$ disjoint cosets of a linear block (n_1, k_1, d_1) code, C_1. Then each codeword in C_3 can be written as $\bar{x}_i + \bar{y}_i + \bar{v}$, with $\bar{v} \in C_1$, where $\bar{x}_i$ is a coset representative of C_2 in C_3 and $\bar{y}_i$ is a coset representative of C_1 in C_2. Let C_4 and C_5 be two linear block (n_4, a_{23}, d_4) and (n_5, a_{12}, d_5) codes, respectively. The linear block $(n_1 + n_4 + n_5, k_3, d_{X3})$ code C_{X3} is defined as

$$C_{X3} \triangleq \{|\bar{x}_i + \bar{y}_i + \bar{v}|\bar{w}|\bar{z}| : \bar{x}_i + \bar{y}_i + \bar{v} \in C_3, \ \bar{w} \in C_4, \ \text{and} \ \bar{z} \in C_5\},$$

and has a minimum distance $d_{X3} = \min\{d_1, d_2 + d_4, d_3 + d_5\}$. A generator matrix of C_{X3} is

$$G_{X3} = \begin{pmatrix} G_1 & 0 & 0 \\ G_2 & G_4 & 0 \\ G_3 & 0 & G_5 \end{pmatrix},$$

where (G_1), $(G_1^\top \quad G_2^\top)^\top$ and $(G_1^\top \quad G_2^\top \quad G_3^\top)^\top$ are the generator matrices of codes C_1, C_2 and C_3, respectively.

Example 78 Let C_1, C_2 and C_3 be $(64, 30, 14)$, $(64, 36, 12)$ and $(64, 39, 10)$ extended BCH codes, respectively, and let C_4 and C_5 be $(7, 6, 2)$ and $(7, 3, 4)$ SPC and maximum-length codes, respectively. Construction X3 results in a $(78, 39, 14)$ code. This code has higher rate (four more information bits) than a shortened $(78, 35, 14)$ code obtained from the extended BCH $(128, 85, 14)$ code.

Generalizations of constructions X and X3 and good families of codes are presented in [Slo, MS, Kas1, Sug, FL3]. The application of these techniques to construct LUEP codes was considered in [Van, MH].

6.2.3 Products of codes

In this section, the important method of code combination known as product, is presented. The simplest method to combine codes is *serially*. That is, the output of a first encoder is taken as the input of a second encoder, and so on. This is illustrated for two encoders, in Figure 53. This is a straightforward method to form of a *product code*. Although simple, this *direct product* method produces very good codes. Very low-rate convolutional codes can be constructed by taking products of binary convolutional codes and block repetition codes.

Example 79 Consider the standard memory-6 rate-1/2 convolutional encoder with generators $(171, 133)$ and minimum distance $d_f = 10$, connected in series with time-sharing of repetition $(2, 1, 2)$ and $(3, 1, 3)$ codes, namely $|(2, 1, 2)|(2, 1, 2)|$ and $|(3, 1, 3)|(3, 1, 3)|$. In other words, every coded bit is repeated two or three times, respectively.

These schemes produce a memory-6 rate-1/4 code and a memory-6 rate-1/6 code, with generators $(171, 171, 133, 133)$ and $(171, 171, 171, 133, 133, 133)$, and $d_f = 20$ and $d_f = 30$, respectively. These codes are optimal [Dho] in the sense that they have the largest free distance

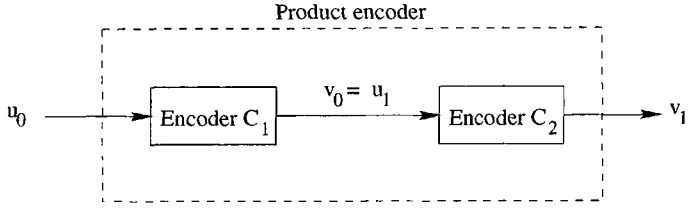


Figure 53 Block diagram of an encoder of a product code.

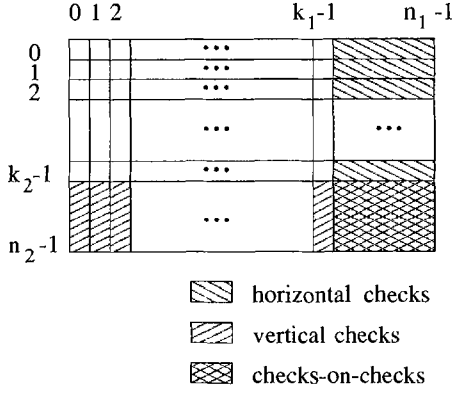


Figure 54 Codeword of a two-dimensional product code.

for a given number of states. This seems to be the first time that they have been expressed in terms of these generators.

However, except for the case of two encoders where the second encoder is the time-sharing of repetition codes, important questions arise when considering a serial connection[3] between two encoders: How should the output of the first encoder be fed into the second encoder? In what follows, let C_1 denote the *outer code* and C_2 denote the *inner code*. Either C_1 or C_2 or both can be convolutional or block codes. If G_1 and G_2 are the generator matrices of the component codes, then the generator matrix of the product code is the Kronecker product, $G = G_1 \otimes G_2$.

In 1954, Elias [Eli1] introduced product or iterated codes. The main idea is as follows. Assume that both C_1 and C_2 are systematic. The codewords of the inner code C_1 are arranged as rows of a rectangular array with n_1 columns, one per code symbol in a codeword of C_1. After k_2 rows have been filled, the remaining $n_2 - k_2$ rows are filled with redundant symbols produced, on a column-by-column basis, by the outer code C_2. The resulting $n_2 \times n_1$ rectangular array is a codeword of the *product code* $C_P \triangleq C_1 \otimes C_2$. Figure 54 depicts the structure of a codeword of a two-dimensional product code. Extension to higher dimensions is straightforward.

---

[3] This is also known as *serial concatenation*. However, in this chapter the term *concatenation* is used with a different meaning.

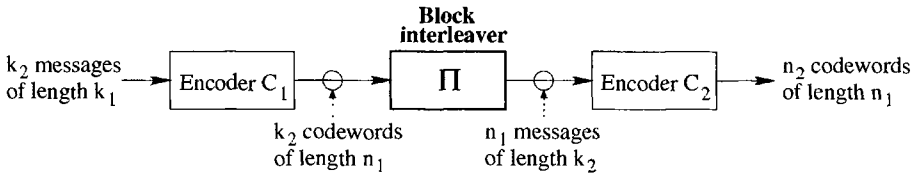


Figure 55 A two-dimensional product encoder with a block interleaver.

	j=0	j=1	j=2	j=3	j=4
i=0	0	2	4	6	8
i=1	1	3	5	7	9

Figure 56 A 2-by-5 block interleaver.

The array codewords are transmitted on a column-by-column basis. With reference to the initial description of a product code, Figure 53, Elias two-dimensional product codes can be interpreted as connecting the two encoders serially, with an *interleaver* in between. This is shown schematically in Figure 55.

As defined by Ramsey [Ram], an interleaver is *a device that rearranges the ordering of a sequence of symbols in some one-to-one deterministic manner.* For the product of linear block codes, naturally, the device is known as a *block interleaver.* The interleaver describes a mapping $m_b(i,j)$ between the elements $a_{i,j}$ in a $k_2 \times n_1$ array, formed by placing k_2 codewords of C_1 as rows, and the elements $u_{m_b(i,j)}$ of an *information vector* $\bar{u} = (\, u_0 \quad u_1 \quad \cdots \quad u_{n_1 n_2 - 1}\,)$.

The one-to-one and onto mapping induced by a $m_1 \times m_2$ block interleaver can also be expressed as a *permutation* $\Pi : i \mapsto \pi(i)$, acting on the set of integers modulo $m_1 m_2$. Writing the array as a one-dimensional vector, $\bar{u}$, by time-sharing of (in that order) the first to the m_1-th row of the array,

$$\bar{u} = (\, u_0 \quad u_1 \quad \cdots \quad u_{m_1 m_2 - 1}\,),$$

the output of the block interleaver is read, via Π, as

$$\bar{u}_\pi \triangleq (\, u_{\pi(0)} \quad u_{\pi(1)} \quad \cdots \quad u_{\pi(m_1 m_2 - 1)}\,), \tag{6.18}$$

where

$$\pi(i) = m_2(i \bmod m_1) + \lfloor \frac{i}{m_1} \rfloor. \tag{6.19}$$

Example 80 Let C_1 and C_2 be linear block SPC $(5,4,2)$ and $(3,2,2)$ codes, respectively. This results in a $(15,8,4)$ product code with the block interleaver shown in Figure 56. The permutation is given by

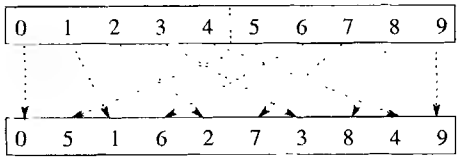
$$\pi(i) = 5(i \bmod 2) + \lfloor \frac{i}{2} \rfloor,$$

and the vector $\bar{u} = (u_0, u_1, u_2, u_3, \cdots, u_9)$ is mapped onto

$$\bar{u}_\pi = (\,(\, u_0 \quad u_5\,) \quad (\, u_1 \quad u_6\,) \quad \cdots \quad (\, u_4 \quad u_9\,)\,) = (\, \bar{u}_0 \quad \bar{u}_1 \quad \cdots \quad \bar{u}_4\,).$$

0	1	2	3	4
5	6	7	8	9

(a)



(b)

Figure 57 (a) Codewords in C_1 as rows; (b) equivalent vector $\bar{u}$ and its permutation u_π.

	j=0	j=1	j=2	j=3	j=4
i=0	0	3	6	9	12
i=1	1	4	7	10	13
i=2	2	5	8	11	14

Figure 58 Mapping $m_b(i,j)$ of a 3-by-5 block interleaver.

This is illustrated in Figure 57. The sub-vectors $\bar{u}_i = (u_i u_{i+5})$, $0 \leq i < 5$, constitute information vectors to be encoded by C_1. Traditionally, codewords are interpreted as two-dimensional arrays,

$$\bar{v} = (\, a_{0,0} \quad a_{1,0} \quad a_{2,0} \quad a_{0,1} \quad a_{1,1} \quad \cdots \quad a_{2,4} \,),$$

where the rows $(\, a_{\ell,0} \quad a_{\ell,1} \quad \cdots \quad a_{\ell,4} \,) \in C_1$, for $\ell = 0, 1, 2$, and the columns $(\, a_{0,\ell} \quad a_{1,\ell} \quad a_{2,\ell} \,) \in C_2$, for $\ell = 0, 1, \cdots 4$. The underlying ordering is depicted in Figure 58. The one-dimensional notation gives the same vector,

$$\bar{v} = (\, (\bar{u}_0, v_0) \quad (\bar{u}_1, v_1) \quad \cdots (\bar{u}_4, v_4) \,),$$

where $(\bar{u}_i, v_i) \in C_2$.

Example 81 Let C_1 and C_2 be two binary SPC $(3,2,2)$ codes. Then C_P is a $(9,4,4)$ code. Although this code has one more redundant bit than an extended Hamming code (or the RM(1,3) code), it can correct errors very easily by simply checking the overall parity of the received rows and columns. Let the all-zero codeword be transmitted over a BSC channel and suppose that the received codeword is

$$\bar{r} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

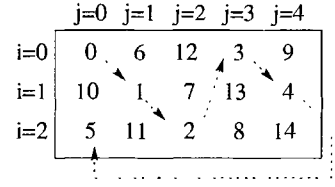


Figure 59 Cyclic mapping m_c for $n_1 = 5, n_2 = 3$.

Recall that the syndrome of a binary SPC $(n, n-1, 2)$ code is simply the sum of the n bits. The second row and the first column will have nonzero syndromes, indicating the presence of an odd number of errors. Moreover, since the other columns and rows have syndromes equal to zero, it is concluded correctly that a single error must have occurred in the first bit of the second row (or the second bit of the first column). Decoding finishes upon complementing the bit in the located error position.

The code in Example 81 above is a member of a family of codes known as *array codes* (see, e.g., [Kas2, BL]). Being product codes, array codes are able to correct bursts of errors, in addition to single errors. Array codes have nice trellis structures [HM], and are related to *generalized concatenated (GC) codes* [HMF], which are the topic of Section 6.2.4.

Let C_i be a linear block (n_i, k_i, d_i) code, $i = 1, 2$. Then the product $C_P = C_1 \otimes C_2$ is a linear block $(n_1 n_2, k_1 k_2, d_P)$ code, where $d_P = d_1 d_2$. In addition, C_P can correct all *bursts of errors* of length up to $b = \max\{n_1 t_2, n_2 t_1\}$, where $t_i = \lfloor (d_i - 1)/2 \rfloor$, for $i = 1, 2$. The parameter b is called the *burst error correcting capability*.

Example 82 Let C_1 and C_2 be two Hamming $(7, 4, 3)$ codes. Then C_P is a $(49, 16, 9)$ code that is capable of correcting up to 4 random errors and bursts of up to 7 errors.

If the component codes are cyclic, then the product code is cyclic [BW]. More precisely, let C_i be a cyclic (n_i, k_i, d_i) code with generator polynomial $g_i(x)$, $i = 1, 2$. Then the code $C_P = C_1 \otimes C_2$ is cyclic if the following conditions are satisfied:

1. The lengths of the codes C_i are relatively prime, i.e., $an_1 + bn_2 = 1$, for two integers a and b;
2. The *cyclic mapping* $m_c(i, j)$ that relates the element $a_{i,j}$ in the rectangular array of Figure 54 with a coefficient $v_{m_c(i,j)}$ of a code polynomial $\bar{v}(x) = v_0 + v_1 + \cdots + v_{n_1 n_2 - 1} x^{n_1 n_2 - 1} \in C_P$, is such that

$$m_c(i, j) = [(j - i) \cdot bn_1 + i] \bmod n_1 n_2, \tag{6.20}$$

for $m_c(i, j) = 0, 1, \cdots, n_1 n_2 - 1$.

When these two conditions are satisfied, the generator polynomial of the cyclic code C_P is given by

$$\bar{g}(x) = \mathrm{GCD}\left(\bar{g}_1(x^{bn_2})\bar{g}_2(x^{an_1}), x^{n_1 n_2} + 1\right). \tag{6.21}$$

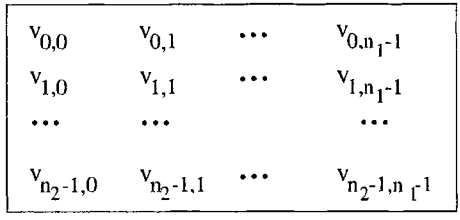


Figure 60 Codeword of an block interleaved code of degree $I = n_2$.

Example 83 An example of the cyclic mapping for $n_1 = 5$ and $n_2 = 3$ is shown in Figure 59. In this case, $(-1)5 + (2)3 = 1$, so that $a = -1$ and $b = 2$. Consequently, the mapping is given by

$$m_c(i, j) = (6j - 5i) \bmod 15.$$

As a check, if $i = 1$ and $j = 2$, then $m_c(1, 2) = (12 - 5) \bmod 15 = 7$; if $i = 2$ and $j = 1$, then $m_c(2, 1) = (6 - 10) \bmod 15 = -4 \bmod 15 = 11$.

The mapping $m_c(i, j)$ indicates *the order in which the digits of the array are transmitted* [BW]. This is not the same as the column-by-column order of the block interleaver for a conventional product code. The mapping described by (6.20) is referred to as a *cyclic interleaver*. Other classes of interleavers are discussed in Section 6.2.4.

With the appearance of turbo codes [BGT] in 1993, there has been intense research activity in novel interleaver structures that perform a pseudo-random arrangement of the codewords of C_1, prior to encoding with C_2. In the next section, interleaved codes are presented. Chapter 8 discusses classes of interleaver structures that are useful in iterative decoding techniques of product codes.

Block interleaved codes

A special case of product code is obtained when the second encoder is the trivial $(n_2, n_2, 1)$ code. In this case, codewords of C_1 are arranged as rows of an n_2-by-n_1 rectangular array and transmitted column-wise, just as in a conventional product code. The value $I = n_2$ is known as the *interleaving degree* [LC] or *interleaving depth*.

The resulting block interleaved code, henceforth denoted as $C_1^{(n_2)}$, can be decoded using the same decoding algorithm of C_1, after reassembling a received word, column-by-column and decoding it row-by-row. Figure 60 shows the schematic of a codeword of an interleaved code, where $(\, v_{i,0} \quad v_{i,1} \quad \cdots \quad v_{i,n_0} \,) \in C_1$, for $0 \leq i < n_2$.

If the error correcting capability of C_1 is $t_1 = \lfloor (d_1 - 1)/2 \rfloor$, then $C_1^{(n_2)}$ can correct any single error burst of length up to $b = t_1 n_2$. This is illustrated in Figure 61. Recall that the transmission order is column by column. If a burst occurs, but it does not affect more than b_1 positions per row, then it can be corrected by C_1. The maximum length of such a burst of errors is n_2 times b_1. Moreover, if code C_1 can already correct (or detect) any single burst of length up to b_1, then $C_1^{(n_2)}$ can correct (or detect) any single burst of length up to $b_1 n_2$.

If C_1 is a cyclic code, then it follows from (6.21) that $C_1^{(n_2)}$ is a cyclic code with generator polynomial $g_1(x^{n_2})$ [PW, LC]. This applies to shortened cyclic codes as well, and the following result holds ([PW], p. 358):

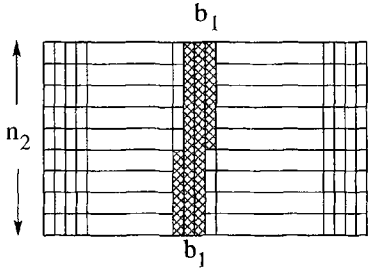


Figure 61 A correctable error burst in a block interleaved codeword.

Interleaving a shortened cyclic (n, k) code to degree ℓ produces a shortened $(n\ell, k\ell)$ code whose burst error correcting capability is ℓ times that of the original code.

Finally, note that the error correcting capability of a product code, $t_P = \lfloor (d_1 d_2 - 1)/2 \rfloor$, can only be achieved if a carefully designed decoding method is applied.

Most of the decoding methods for product codes use a *two-stage decoding* approach. In the first stage, an errors-only algebraic decoder for the row code C_1 is used. Then *reliability weights* are assigned to the decoded symbols, based on the number of errors corrected. The more errors are corrected, the less reliable the corresponding estimated codeword $\hat{v}_1 \in C_1$ is.

In the second stage, an errors-and-erasures algebraic decoder for the column code C_2 is used, with an increasing number of erasures declared in the *least reliable positions* (those positions for which the reliability weights are the smallest), until a sufficient condition on the number of corrected errors is satisfied. This is the approach originally proposed in [RR, Wel]. The second decoding stage is usually implemented with the GMD algorithm, which is discussed in Section 7.6. More on decoding of product codes can be found in Chapter 8.

6.2.4 *Concatenated codes*

In 1966, Forney [For1] introduced a clever method of combining two codes, called *concatenation*. The scheme is illustrated in Figure 62. Concatenated codes[4] that are based on outer Reed-Solomon codes and inner convolutional codes have been to date[5] perhaps the most popular choice of ECC schemes for digital communications. In general, the outer code, denoted C_1, is a nonbinary linear block (N, K, D) code over $GF(2^k)$. The codewords of C_1 are stored in an interleaver memory. The output bytes read from the interleaver are then passed through an encoder for an inner code, C_2. The inner code C_2 can be either a block code or a convolutional code. When block codes are considered, and C_2 is a binary linear block (n, k, d) code, the encoder structure is shown in Figure 62. Let $C = C_1 \star C_2$ denote the concatenated code with C_1 as the outer code and C_2 as the inner code. Then C is a binary linear block (Nn, Kk, Dd) code.

The purpose of the interleaver between the outer and the inner code is two-fold. First, it serves to convert the bytes of size k into vectors of the same dimension (number of information bits) as the inner code, be it binary or nonbinary, a linear block (n, k', d) code or a rate-

[4] Also referred to by some authors as *cascaded* codes.

[5] Before the arrival of turbo codes and LDPC codes.

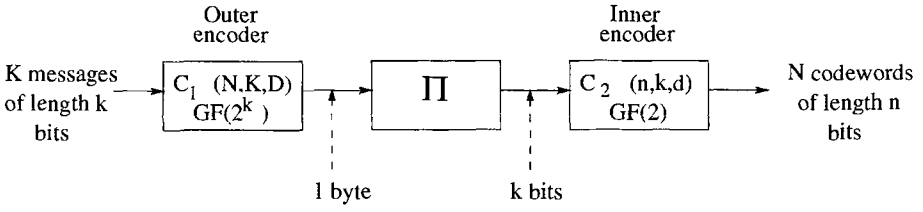


Figure 62 An encoder of a concatenated code.

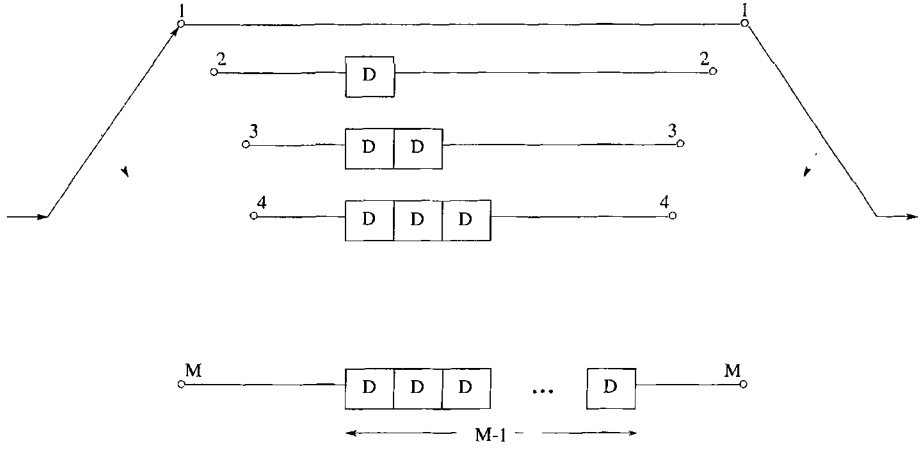


Figure 63 A convolutional interleaver.

k'/n convolutional code, for which in general $k' \neq k$. On the other hand, as discussed in the previous section, interleaving allows breaking of bursts of errors. This is useful when concatenated schemes with inner convolutional codes are considered, because the Viterbi decoder tends to produce bursts of errors [CY, Mor]. There are several types of interleavers that are used in practice. The most popular appears to be the *convolutional interleaver* [For5], which is a special case of a *Ramsey interleaver* [Ram]. The basic structure of a convolutional interleaver is shown in Figure 63. The deinterleaver structure is identical, with the exception that the switches are initially in position M and rotate in the opposite direction.

An important advantage of concatenated codes (and product codes) is that decoding can be based on the decoding of each component code. This results in a dramatic reduction in complexity, compared to a decoder for the entire code.

Example 84 Let C_1 be a $(7,5,3)$ RS code[6] with zeros $\{1,\alpha\}$, where α is a primitive element of $GF(2^3)$, and $\alpha^3+\alpha+1=0$. Let C_2 be the maximum-length-sequence $(7,3,4)$ code of Example 77. Then $C = C_1 \star C_2$ is a binary linear block $(49,15,12)$ code. This code has six information bits less than a shortened $(49,21,12)$ code obtained from the extended BCH $(64,36,12)$ code. However, it is simpler to decode. Let $\bar{v}(x) = (x^4+\alpha^4)\bar{g}(x) = \alpha^5 + x + \alpha^4 x^2 + \alpha x^4 + \alpha^3 x^5 + x^6$ be a codeword in the RS $(7,5,3)$ code, where $\bar{g}(x) = x^2 + \alpha^3 x + \alpha$.

[6] RS codes are the topic of Chapter 4.

α^5	1	α^4	0	α	α^3	1
1	0	1	0	0	0	0
1	0	1	0	1	1	0
1	1	0	0	0	1	1
0	1	1	0	0	1	1
0	1	1	0	1	0	1
0	0	0	0	1	1	0
1	1	0	0	1	0	1

Figure 64 A codeword in the concatenated code $C_1 \star C_2$, with C_1 the RS$(7, 5, 3)$ code over $GF(2^3)$ and C_2 a binary cyclic $(7, 3, 4)$ code.

Using the table on page 41, the elements of $GF(2^3)$ can be expressed as vectors of 3 bits. A 3-by-7 array whose columns are the binary vector representations of the coefficients of the code polynomial $\bar{v}(x)$ is obtained. Then encoding by the generator polynomial of C_2 is applied to the columns to produce 4 additional rows of the codeword array. For clarity, the following systematic form of the generator matrix of C_2 is used, which is obtained after exchanging the third and sixth columns of G in Example 77,

$$G' = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

Figure 64 shows the codeword array corresponding to $\bar{v} \in C_1$.

6.2.5 *Generalized concatenated codes*

In 1974, Blokh and Zyablov [BZ] and Zinov'ev [Zin] introduced the powerful class of generalized concatenated (GC) codes. This is a family of error correcting codes that can correct both random errors and random bursts of errors. As the name implies, GC codes generalize Forney's concept of concatenated codes, by the introduction of a *subcode hierarchy* (or subcode partition) of the inner code C_I and several outer codes, one for each *partition level*. The GC construction combines the concepts of direct-sum, or coset decomposition, and concatenation. Before defining the codes, some notation is needed.

A linear block (n, k, d) code C is said to be *decomposable* with respect to its linear block (n, k_i, d_i) subcodes C_i, $1 \leq i \leq M$, if the following conditions are satisfied:

(Sum) $C = C_1 + C_2 + \cdots + C_M$;

(D) For $\bar{v}_i \in C_i$, $1 \leq i \leq M$, $\bar{v}_1 + \bar{v}_2 + \cdots + \bar{v}_M = \bar{0}$ if and only if $\bar{v}_1 = \bar{v}_2 = \cdots = \bar{v}_M = \bar{0}$.

For $1 \leq i \leq M$, let C_{Ii} be a linear block (n_I, k_{Ii}) code over $GF(q)$ such that

($\mathrm{D_I}$) for $\bar{u}_i \in C_{Ii}$ with $1 \leq i \leq M$, $\bar{u}_1 + \bar{u}_2 + \cdots + \bar{u}_M = \bar{0}$, if and only if $\bar{u}_i = \bar{0}$ for $1 \leq i \leq M$.

Let δ_i denote the minimum Hamming distance of the direct-sum code $C_{Ii}+C_{Ii+1}+\cdots+C_{IM}$. Let C_{Oi} be a linear block (n_O, k_{Oi}, d_{Oi}) code over $GF(q^{k_{Ii}})$ and let $C_i^\star = C_{Oi} \star C_{Ii}$. The generalized concatenated code C is defined as the direct-sum

$$C \triangleq C_1^\star + C_2^\star + \cdots + C_M^\star. \tag{6.22}$$

Then, the condition (D) on C follows from the condition (D_I). The minimum Hamming distance d of C is lower bounded [TYFKL] as

$$d \geq \min_{1 \leq i \leq M} \delta_i d_{Oi}. \tag{6.23}$$

As with (single level) concatenated codes, a main advantage of GC codes is that multi-stage decoding up to the distance given by the right-hand side of (6.23) is possible [TYFKL, MFKL].

Unequal error protection

Another advantage of this class of codes is that it is relatively easy to coordinate the distances of the outer and inner codes to obtain linear block or convolutional codes with *unequal error protection* (UEP) capabilities. If the direct-sum conditions above are satisfied, and in addition the products of the minimum distances satisfy the following inequalities,

$$\delta_1 d_{O1} \geq \delta_2 d_{O2} \geq \cdots \geq \delta_M d_{OM}, \tag{6.24}$$

then codewords in correspondence with $k_{Oi}k_{Ii}$ symbols over $GF(q)$ will have an error correcting capability, $\lfloor(\delta_i d_{Oi} - 1)/2\rfloor$, that decreases with the level i, for $1 \leq i \leq M$. As a result, the messages encoded in the top (low values of i) partition levels will have enhanced error correcting capabilities, compared to those associated with the lowest partition levels. Constructions of this type are reported in [DYS, MH].

A construction

Let a linear block (n_I, k_1, d_1) code C_1 over $GF(q)$ be *partitioned* into a chain of M (n_I, k_i, d_i) subcodes C_i, $i = 2, 3, \cdots, M+1$, such that

$$C_1 \supset C_2 \supset \cdots \supset C_{M+1},$$

where, for convenience, $C_{M+1} \triangleq \{\bar{0}\}$, and $d_{M+1} \triangleq \infty$.

Let $C_{Ii} = [C_i/C_{i+1}]$ denote a linear block (n_I, k_{Ii}, δ_i) subcode of C_i, which is a *set of coset representatives* of C_{i+1} in C_i, of dimension $k_{Ii} = k_i - k_{i+1}$ and minimum Hamming distance $\delta_i \geq d_i$. Then C_1 has the following *coset decomposition* [For6]

$$C_1 = C_{I1} + C_{I2} + \cdots + C_{IM}. \tag{6.25}$$

Let C_{Oi} denote a linear block (n_O, k_{Oi}, d_{Oi}) code C_{Oi} over $GF(q^{k_{Ii}})$, where $k_{Ii} = \dim(C_i/C_{i+1}) = k_i - k_{i+1}$, $i = 1, 2, \cdots, M$. Then the direct sum of concatenated codes

$$C = C_{O1} \star C_{I1} + C_{O2} \star C_{I2} + \cdots + C_{OM} \star C_{IM}.$$

is an $(n_O n_I, k, d)$ linear block code of dimension and minimum Hamming distance, respectively [BZ],

$$k = \sum_{i=1}^{M} k_{Ii} k_{Oi}, \qquad \text{and} \qquad d \geq \min_{1 \leq i \leq M} \{\delta_i d_{Oi}\}. \tag{6.26}$$

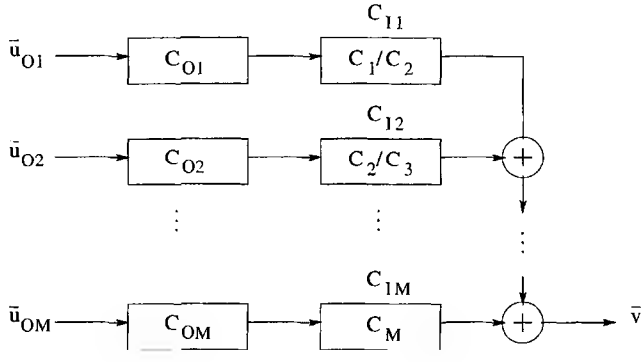


Figure 65 Encoder structure of a generalized concatenated code with M partition levels.

Note that equality holds in (6.26) when C_{I_i}, $1 \leq i \leq M$, contains the all-zero codeword.

As for the choice of component codes. Note that, since in general the dimensions of the coset representatives are distinct, the outer codes can be selected as RS codes or shortened RS codes.

Binary RM codes are good candidates as inner codes, because they have the following *subcode property*,

$$\mathrm{RM}(r,m) \subset \mathrm{RM}(r+1,m),$$

for $0 \leq r < m$, and $m \geq 2$.

Example 85 Consider the r-th order RM codes of length 8, $\mathrm{RM}(r,3)$. Then

$$\mathrm{RM}(3,3) \supset \mathrm{RM}(2,3) \supset \mathrm{RM}(1,3) \supset \mathrm{RM}(0,3) \supset \{\bar{0}\},$$

and it follows from (6.25) that

$$\mathrm{RM}(3,3) = [\mathrm{RM}(3,3)/\mathrm{RM}(2,3)]+[\mathrm{RM}(2,3)/\mathrm{RM}(1,3)]+[\mathrm{RM}(1,3)/\mathrm{RM}(0,3)]+\mathrm{RM}(0,3).$$

This decomposition can be also appreciated from the generator matrix of $\mathrm{RM}(3,3)$, expressed as

$$G = \begin{pmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix},$$

where G_i has been defined as the generator matrix of the set of coset representatives of $\mathrm{RM}(3-i,3)$ in $\mathrm{RM}(3-i+1,3)$, or $[\mathrm{RM}(3-i+1,m)/\mathrm{RM}(3-i,m)]$, for $i = 1,2,3$, and where G_4 is the generator matrix of $\mathrm{RM}(0,3)$.

According to this partition of RM$(3,3)$, a GC code can be designed with up to four levels. Note that the outer codes should be over $GF(2)$, for the first and fourth partition levels, and over $GF(2^3)$ for the second and third partition levels.

Also, some of the subcodes of RM$(3,3)$ themselves can be used as inner codes to obtain a GC code with a reduced number of levels. If RM$(2,3)$ is used as the inner code of a GC code, then the number of partition levels is three, as the generator matrix of RM$(2,3)$ is obtained from that of RM$(3,3)$ by removing G_1 (the top row).

Let RM$(2,3)$ be selected as the inner code, and an RS $(8,1,8)$ code C_{O1} over $GF(2^3)$, an RS $(8,5,4)$ code C_{O2} over $GF(2^3)$, and a binary linear SPC $(8,7,2)$ code C_{O3} be selected as outer codes. This gives a binary linear GC $(64,25,16)$ code which has one more information bit than the binary extended BCH $(64,24,16)$ code.

It is now shown how Reed-Muller codes can be expressed as GC codes. Recall from Section 6.2.2, Equation (6.16) on page 108, that the $(r+1)$-th order RM code of length 2^{m+1} can be expressed as $\mathrm{RM}(r+1,m+1) = |\mathrm{RM}(r+1,m)|\mathrm{RM}(r+1,m)+\mathrm{RM}(r,m)|$. Let $G(r,m)$ denote the generator matrix of RM(r,m). Then

$$\begin{aligned} G(r+1,m+1) &= \begin{pmatrix} G(r+1,m) & G(r+1,m) \\ 0 & G(r,m) \end{pmatrix} \\ &= G(r+1,m)\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} + G(r,m)\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}. \end{aligned} \tag{6.27}$$

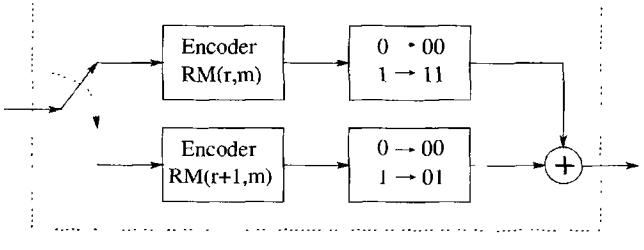


Figure 66 An encoder for the binary $RM(r+1,m+1)$ code when viewed as a two-level GC code.

This expression holds for each r-th order RM code, with $r \geq 1$ and $m > 1$, so that a recursion is obtained. From (6.27) it follows that RM$(r+1,m+1)$ is a GC code with inner codes the $(2,2,1)$ code and its partition into cosets of the repetition $(2,1,2)$ code, and outer codes RM(r,m) and RM$(r+1,m)$, respectively. An encoder for RM$(r+1,m+1)$ is shown in Figure 66. This recursive structure of RM codes is advantageous when designing soft-decision decoding algorithms for RM codes[7], which can be based on simple repetition and parity-check codes. This is precisely what was done in [SB].

[7] Soft-decision decoding is presented in Chapter 7.

7

Soft-decision decoding

In this chapter, decoding with *soft information* from the channel is considered. For simplicity of exposition, binary transmission over an AWGN channel is assumed. In Chapter 9, the general case of multilevel transmission is discussed. To provide a motivation for the use of soft decision decoding, it can be argued that the noise environment in data retrieval or signal reception is continuous in nature, not discrete. This means that the received symbols are (quantized) *real numbers* (voltages, currents, etc.), and not binary or $GF(2^m)$ symbols.

When making hard decisions on the received symbols, on a symbol-by-symbol basis, errors may be introduced. This is illustrated in Figure 67.

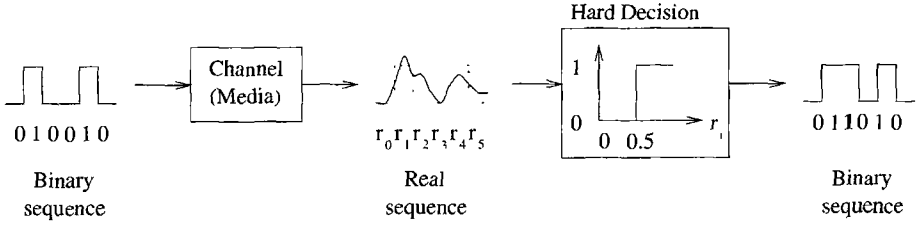


Figure 67 Example of an error introduced by hard-decision decoding.

Basically, there are two methods of decoding an error correcting code, based on a received real-valued sequence:

1. Hard-decision decoding (HDD):
   When hard decisions are made on the channel values, errors are introduced. The goal of HDD is to correct binary errors induced in the hard-decision process. The first part of the book was devoted to different techniques of HDD for linear block codes, cyclic codes and convolutional codes.
2. *Soft-decision decoding (SDD)*:
   In this case, received values from the channel are directly processed by the decoder in order to estimate a code sequence. A special case of SDD is maximum-likelihood decoding (MLD), in which case the *most-likely* code sequence is output by the decoder. An important point to keep in mind here is that SDD needs to know the statistics of the channel noise.

In general, SSD is computationally more intensive than HDD. There are two main reasons for this. One is that SDD needs to operate on real numbers. In a practical application, these

numbers will be *quantized* to a finite number of bits. In some binary transmission systems over an AWGN channel, it is known that 8 quantization levels, or 3 bits, are adequate to achieve practically the same performance as with unquantized channel values [Mas3, OCC].

The second reason for the increased complexity in SDD is that the *a-posteriori* statistics of the coded symbols, given the received values, need to be computed. However, in return for the increase in implementation cost there is potentially much better performance to be attained. As shown in Chapter 1, with binary transmission over an AWGN channel, to get the same error performance, *SDD requires 2 to 3 dB less* signal-to-noise power ratio than HDD. This is to say that, with SDD, the transmitted power can be 50% to 63% lower compared to HDD which translates into smaller transmit antennas or, alternatively, smaller receive antennas for the same transmission power.

7.1 Binary transmission over AWGN channels

For AWGN channels, the *a-posteriori* probability of the received values, r_i, given that code symbols v_i are sent, is given by

$$p(r_i|v_i) = p_{n_i}(r_i - m(v_i)) = \frac{1}{\sqrt{\pi N_0}} e^{-(r_i - m(v_i))^2/N_0} \tag{7.1}$$

As shown in Chapters 1 and 5, the MLD metrics become the *squared Euclidean distances*, $D^2(r_i, m(v_i)) = (r_i - m(v_i))^2$. Let E denote the energy of a transmitted symbol. The mapping rule or modulation employed is BPSK[1],

$$m(v_i) = \begin{cases} \sqrt{E} & \text{if } v_i = 0; \\ -\sqrt{E} & \text{if } v_i = 1. \end{cases} \tag{7.2}$$

which can be expressed as $m(v_i) = (-1)^{v_i}\sqrt{E}$. With BPSK transmission over an AWGN channel, decoding metrics for binary transmission can be simplified by noticing that if $v_i = 1$ is transmitted then

$$p(r_i|1) = p_{n_i}(r_i + 1) = \frac{1}{\sqrt{\pi N_0}} e^{-(r_i+1)^2/N_0}.$$

Taking the natural logarithm and removing constant terms, it follows that the *log-likelihood metric* $-(r+1)^2$ is proportional to $-r$. If $v_i = 0$ then the metric is proportional to r. Therefore, the following is important to note:

With binary transmission over an AWGN channel, metric computation is reduced to changing the signs of the received values.

7.2 Viterbi algorithm with Euclidean metric

The VD can be used to decode convolutionally coded binary data, BPSK modulated, and transmitted over an AWGN channel. With respect to the hard-decision VD algorithm, studied in Section 5.4, two changes are required:

[1] Binary phase-shift keying.

1. Branch-metric generator (BMG) stage
   For an AWGN channel, as shown in the previous section, the metric is proportional to the correlation between the received and the candidate code sequence. Therefore, instead of Euclidean distances, *correlation* metrics can be used.
2. Add-compare-select (ACS) stage:
   Instead of minimizing the distance, the VD seeks to *maximize* the correlation metric.

Example 86 Consider a memory-2 rate-1/2 convolutional encoder with generators (7,5). Suppose that the information sequence is $\bar{u} = (110100)$. The corresponding path is shown in Figure 68.

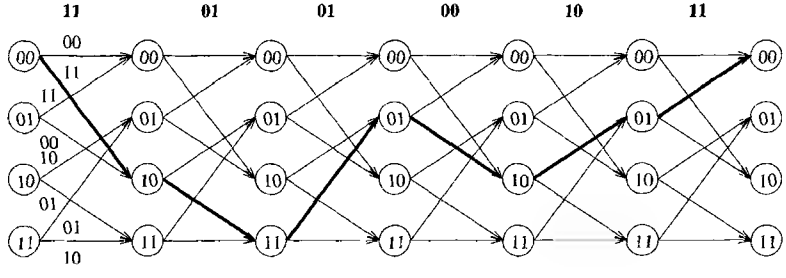


Figure 68 A path in the trellis corresponding to $\bar{u} = (110100)$.

The transmitted sequence (normalized with respect to $\sqrt{E}$) is

$$t(\bar{v}) = (-1, -1, 1, -1, 1, -1, 1, 1, -1, 1, -1, -1).$$

Let the received sequence, after transmission over an AWGN channel and quantization to 3 bits, be:

$$\bar{r} = (-4, -1, -1, -3, +2, -3, +3, +3, -3, +3, -3, +1).$$

Note that the hard-decision received sequence (given by the sign bits) is:

$$\bar{r}_H = (11, \underline{1}1, 01, 00, 10, 1\underline{0}),$$

so that the hard-decision detection process introduces two bit errors. The operation of the Viterbi decoder is illustrated in Figures 69 to 74. The evolution of the metric values with respect to the decoding stages is shown in the following table:

State/Stage	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
$S_i^{(0)}$	0	+5	+7	+6	+12	+14	+26
$S_i^{(1)}$	0	+3	+5	+12	+14	+24	+18
$S_i^{(2)}$	0	+5	+9	+8	+18	+14	+22
$S_i^{(3)}$	0	+3	+7	+14	+14	+20	+24

The decoded information sequence is $\hat{u} = (1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0)$, and two errors have been corrected.

All the implementation issues related to Viterbi decoding, discussed in Sections 5.4.3 and 5.5.1, apply in the case of soft-decision. In particular, metric normalization must be carefully taken into account.

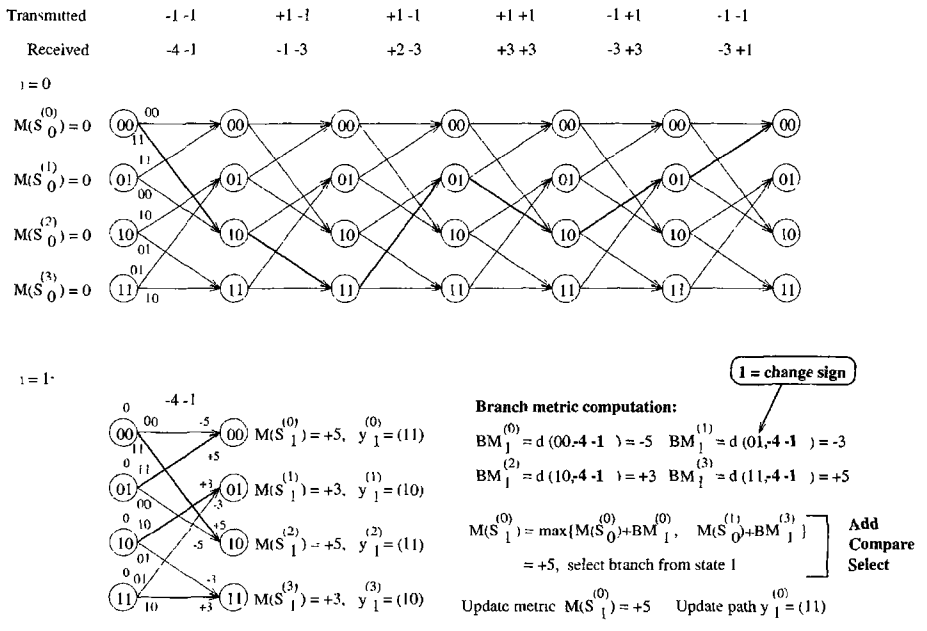

Transmitted -1 -1 +1 -1 +1 -1 +1 +1 -1 +1 -1 -1
Received -4 -1 -1 -3 +2 -3 +3 +3 -3 +3 -3 +1
i = 0
M(S0(0)) = 0
M(S0(1)) = 0
M(S0(2)) = 0
M(S0(3)) = 0
i = 1:
-4 -1
M(S1(0)) = +5, y1(0) = (11)
M(S1(1)) = +3, y1(1) = (10)
M(S1(2)) = +5, y1(2) = (11)
M(S1(3)) = +3, y1(3) = (10)
1 = change sign
Branch metric computation:
BM1(0) = d (00,-4 -1) = -5 BM1(1) = d (01,-4 -1) = -3
BM1(2) = d (10,-4 -1) = +3 BM1(3) = d (11,-4 -1) = +5
M(S1(0)) = max{M(S0(0))+BM1(0), M(S0(1))+BM1(3)}
= +5, select branch from state 1
Add
Compare
Select
Update metric M(S1(0)) = +5 Update path y1(0) = (11)

Figure 69 Soft-decision Viterbi decoder operation at $i = 1$.

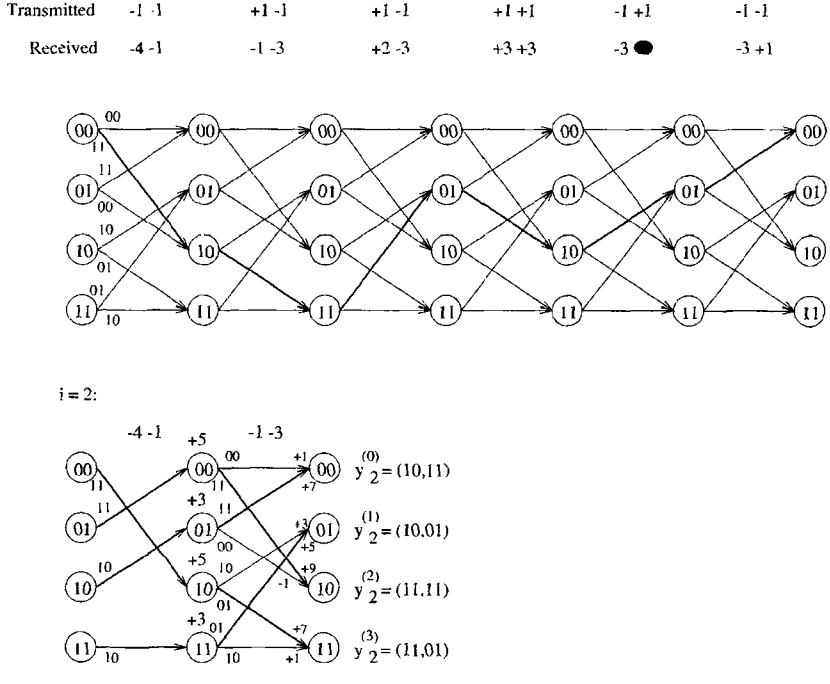
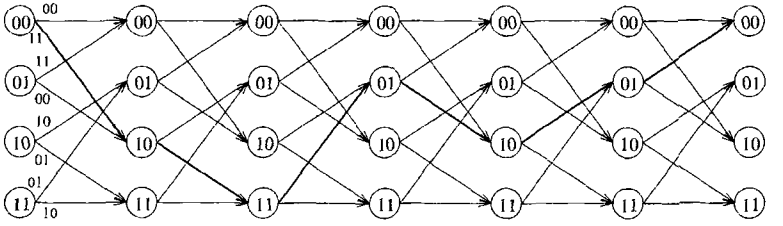


Figure 70 Soft-decision Viterbi decoder operation at $i = 2$.

Transmitted	-1 -1	+1 -1	+1 -1	+1 +1	-1 +1	-1 -1
Received	-4 -1	-1 -3	+2 -3	+3 +3	-3 +3	-3 +1



$i = 3$:

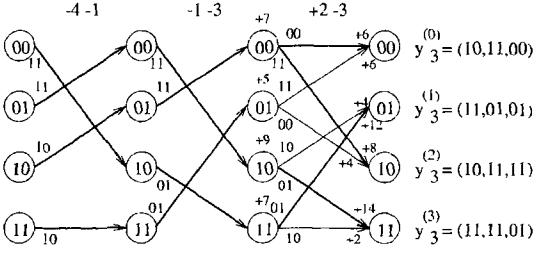
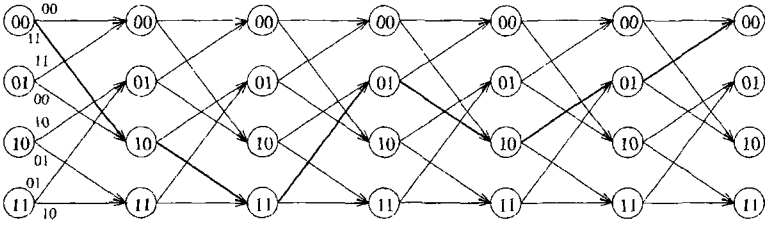


Figure 71 Soft-decision Viterbi decoder operation at $i = 3$.

Transmitted	-1 -1	+1 -1	+1 -1	+1 +1	-1 +1	-1 -1
Received	-4 -1	-1 -3	+2 -3	+3 +3	-3 +3	-3 +1



$i = 4$:

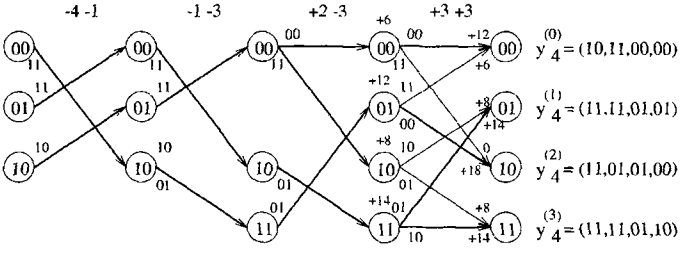
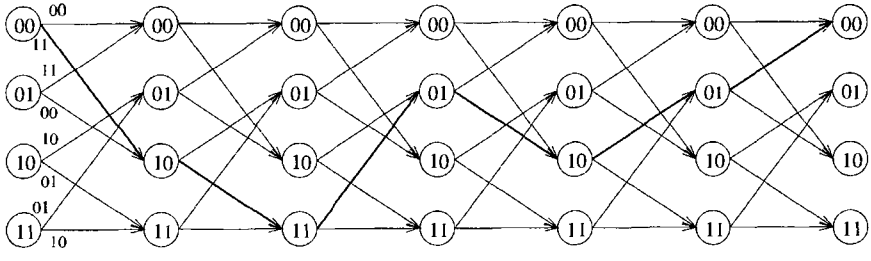


Figure 72 Soft-decision Viterbi decoder operation at $i = 4$.

Transmitted	-1 -1	+1 -1	+1 -1	+1 +1	-1 +1	-1 -1
Received	-4 -1	-1 -3	+2 -3	+3 +3	-3 +3	-3 +1



i = 5:

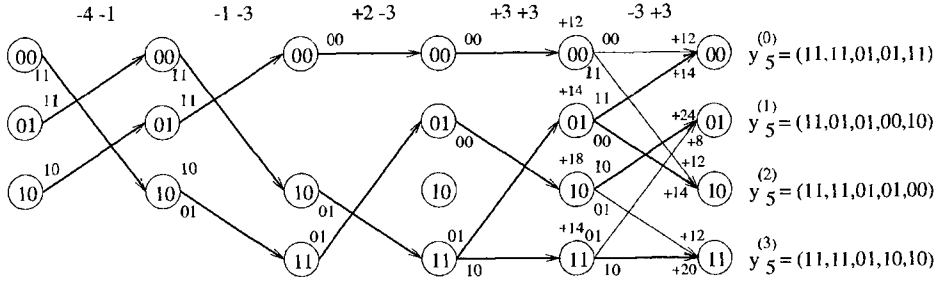


Figure 73 Soft-decision Viterbi decoder operation at $i = 5$.

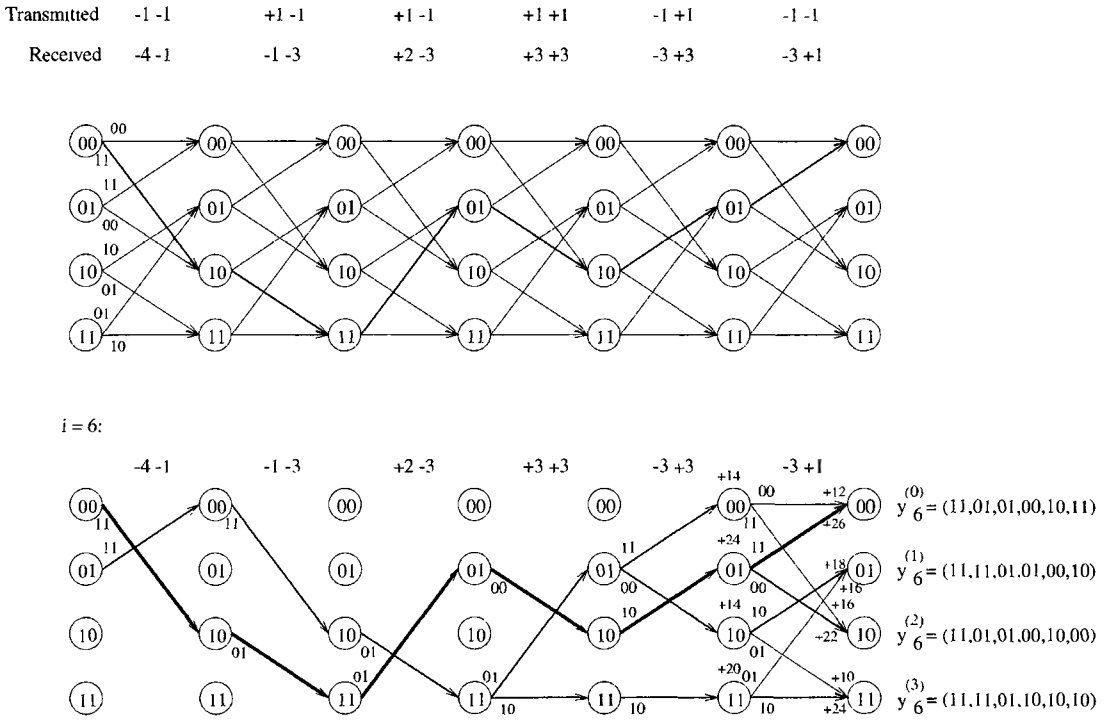


Figure 74 Soft-decision Viterbi decoder operation at $i = 6$.

7.3 Decoding binary linear block codes with a trellis

The Viterbi algorithm can also be applied to linear block codes. A *syndrome trellis* for a binary linear (N, K) block code C can be constructed from its parity-check matrix as follows [Wol]. Let $(v_1, v_2, \cdots, v_N)$ denote a codeword of C. At time i, $1 \leq i \leq N$, states in the trellis are specified by the *partial syndromes*:

$$\bar{s}_i = \sum_{j=1}^{i} v_j \bar{h}_j, \tag{7.3}$$

where the sum is over $GF(2)$, $\bar{h}_j$ is the j-th column of H, and $\bar{s}_0^{\top} \triangleq (0\ 0\ \cdots\ 0)$. The maximum number of states in the syndrome trellis is $\min\{2^K, 2^{N-K}\}$. The syndrome trellis has the property that it has the smallest possible number of states. A trellis satisfying this condition is said to be a *minimal trellis*.

Example 87 Consider a binary cyclic Hamming $(7, 4, 3)$ code with $\bar{g}(x) = x^3 + x + 1$. Then $\bar{h}(x) = 1 + x + x^2 + x^4$, and a parity-check matrix for C is

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

To label a state $\bar{s}_j^{\top} = (s_0\ s_1\ s_2)$ an integer I_j is assigned, such that

$$I_j = s_0 + s_1 \times 2 + s_2 \times 2^2.$$

The trellis has maximum of 8 states (since $2^{n-k} = 2^3$) at times $i = 3$ and $i = 4$, and is shown in Figure 75.

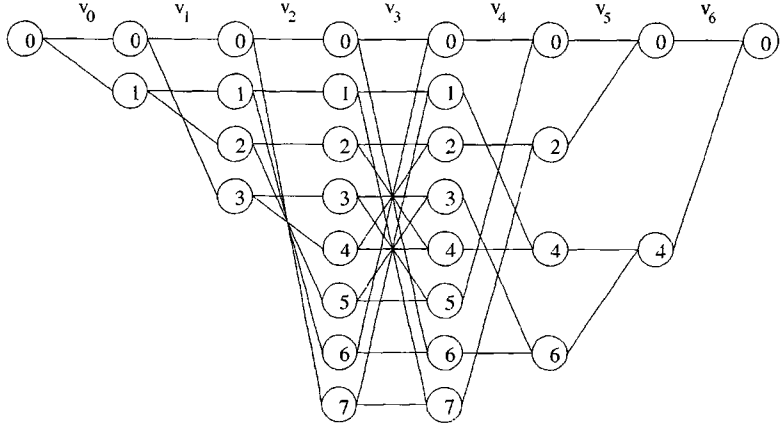


Figure 75 Trellis-structure of a Hamming $(7, 4, 3)$ code.

It is interesting to note that with a syndrome trellis there is no need to label the branches with the coded bits. A transition between two states with the same label corresponds to coded bit 0, as seen from Equation (7.3): If the coded bit is 0, then the sum does not change.

Some observations about the trellis structure of linear block codes are the following: In general, the trellis of a binary cyclic code has an irregular structure. As a result, implementation of the Viterbi algorithm will be more intricate than in the case of convolutional codes.

For some classes of codes, such as extended[2] BCH codes and Reed-Muller codes, the trellis can be divided into sections. This results in a more regular and symmetric trellis structure with many parallel subtrellises, which may be used to build very high-speed Viterbi decoders for block codes [LKFF, HM].

7.4 The Chase algorithm

The Chase algorithm [Cha] is a suboptimal decoding procedure that uses a *set or list of most likely error patterns*. These error patterns are selected based on the *reliability* of the received symbols. Each error pattern is added to the hard-decision received word and decoded using a *hard-decision decoder*. Each decoded codeword is scored by computing its *metric* with respect to the received (soft-decision) sequence. The codeword with the *best metric* is selected as the most likely.

Let C be a binary linear (N, K, d) block code, capable of correcting any combination of $t = \lfloor (d-1)/2 \rfloor$ or less random bit errors. Let $\bar{r} = (r_1, r_2, \cdots, r_N)$ be the received word from the output of the channel, $r_i = (-1)^{c_i} + w_i$, where w_i is a zero-mean Gaussian random variable with variance $N_0/2$, $i = 1, 2, \cdots, N$. The sign bits of the received values represent the hard-decision received word,

$$\bar{z}_0 = (\, z_{0,0} \quad z_{0,1} \quad \cdots \quad z_{0,N-1}\,), \qquad z_{0,j} = \operatorname{sgn}(r_j),\ 0 \le j < N, \tag{7.4}$$

where

$$\operatorname{sgn}(x) = \begin{cases} 0, & \text{if } x \ge 0; \\ 1, & \text{otherwise.} \end{cases}$$

The *reliabilities* of the received channel values, for binary transmission over an AWGN channel, are the amplitudes $|r_i|$. The received symbol reliabilities are ordered with a sorting algorithm (e.g., *quick sort*). The output of the algorithm is a list of indexes I_j, $j = 1, 2, \cdots, N$, such that

$$|r_{I_1}| \le |r_{I_2}| \le \cdots \le |r_{I_N}|.$$

In the first round of decoding, the hard-decision received word $\bar{z}_0$ is fed into a hard-decision decoder. Let $\bar{v}_0$ denote the decoded codeword. Then the metric of $\bar{v}_0$ with respect to the received word $\bar{r}$

$$\nu_0 = \sum_{j=1}^{N} (-1)^{v_{0j}} \times r_j, \tag{7.5}$$

is computed and its value stored as the maximum.

Chase classified his algorithm into three types, according to the error pattern generation.

- **Type-I**
  Test all error patterns within radius $(d-1)$ from the received word.

---

[2] Extending codes is covered in Chapter 6.

- **Type-II**
  Test the error patterns with at most $\lfloor (d-1)/2 \rfloor$ errors located outside the bits having the $\lfloor d/2 \rfloor$ lowest reliabilities. Compared to Type-I, the performance of Chase Type-II algorithm is only slightly inferior, while at the same time having significantly reduced number of test patterns.
- **Type-III**
  Test those error patterns with i ones in the i least reliable bit positions, with i odd and $1 \leq i \leq d-1$. It should be noted that this algorithm is closely related to the GMD decoding algorithm (see also [ML], p. 168). GMD decoding of RS codes is the topic of Section 7.6.

Because of its reduced complexity and good performance, among the three types above, Chase algorithm Type-II is the most popular and described next. A flow diagram of Chase algorithm is shown in Figure 76.

Chase type-II algorithm

- For $i = 1, 2, ..., 2^t - 1$, an error pattern $\bar{e}_i$ is added to the hard-decision received word: $\bar{z}_i = \bar{e}_i \oplus \bar{z}_0$.
- The error patterns are generated among the t least reliable positions. That is, positions $\{I_1, I_2, \cdots, I_t\}$, for which reliabilities (amplitudes) are the smallest.
- Each $\bar{z}_i$ is input to a hard-decision decoder, producing a codeword $\bar{v}_i$, $i = 1, 2, ..., 2^t - 1$.
- The metric is computed according to Equation (7.5) and if maximum, codeword $\bar{v}_i$ stored as the most likely.

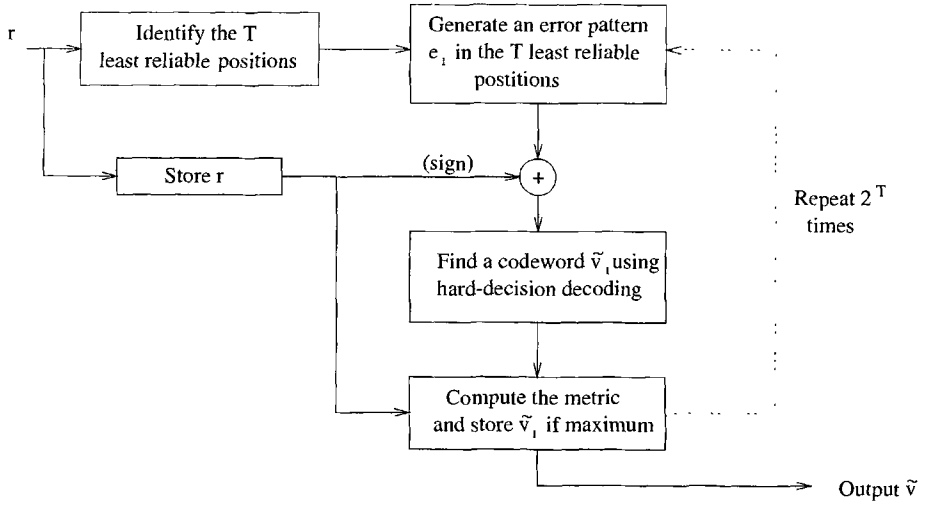

r
Identify the T least reliable positions
Generate an error pattern e_i in the T least reliable postitions
Store r
(sign)
+
Repeat 2^T times
Find a codeword ṽ_i using hard-decision decoding
Compute the metric and store ṽ_i if maximum
Output ṽ

Figure 76 Flow diagram of Chase type-II decoding algorithm.

If desired, the algorithm can output a list of codewords. This is a useful feature in producing soft-outputs for use of the Chase algorithm in iterative decoding of block turbo codes, see Section 8.2.3.

7.5 Ordered statistics decoding

The ordered statistics decoding (OSD) algorithm [FL1] is similar to the Chase algorithm in the sense of creating a list of error patterns and using hard-decision decoding. However, unlike the algorithm in the previous section, which deals with *least reliable code positions*, OSD considers a list of codewords drawn from a set of (permuted) *most reliable information positions*.

Assume that an (N, K) linear block code C, with generator matrix G and minimum distance $d_H \geq 2t + 1$, is used with binary transmission over an AWGN channel. Let $\bar{c} = (c_1, c_2, \cdots, c_N)$ denote a codeword of C, and let $\bar{r} = (r_1, r_2, \cdots, r_N)$ be the received sequence.

As in the Chase algorithm above, the decoding process begins by *reordering* the components of the received sequence in decreasing order of reliability value. Let

$$\bar{y} = (y_1, y_2, \cdots, y_N)$$

denote the resulting sequence, where $|y_1| \geq |y_2| \geq \cdots \geq |y_N|$. This reordering defines a *permutation mapping* λ_1 such that $\bar{y} = \lambda_1[\bar{r}]$.

The next step is to permute the columns of G using λ_1:

$$G' = \lambda_1[G] = (\bar{g}'_1 \quad \bar{g}'_2 \quad \cdots \quad \bar{g}'_N),$$

where $\bar{g}'_j$ denotes the j-th column of G'.

A *most reliable basis* is obtained in the next step of the algorithm as follows:

Starting from the first column of G', find the first K linearly independent (LI) columns with the largest associated reliability values. Then, these K LI columns are used as the first K columns of a new matrix G'', maintaining their reliability order. The remaining $(N - K)$ columns of G'' are also arranged in decreasing reliability order. This process defines a second permutation mapping λ_2, such that

$$G'' = \lambda_2[G'] = \lambda_2[\lambda_1[G]].$$

Applying the map λ_2 to $\bar{y}$, results in the reordered received sequence:

$$\bar{z} = \lambda_2[\bar{y}] = (z_1, z_2, \cdots, z_k, z_{k+1}, \cdots, z_N),$$

with $|z_1| \geq |z_2| \geq \cdots \geq |z_K|$, and $|z_{k+1}| \geq \cdots \geq |z_N|$. By performing elementary row operations on G'', a systematic form G_1 can be obtained:

$$G_1 = (I_k \quad P) = \begin{pmatrix} 1 & 0 & \cdots & 0 & p_{1,1} & \cdots & p_{1,N-K} \\ 0 & 1 & \cdots & 0 & p_{2,1} & \cdots & p_{2,N-K} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{k,1} & \cdots & p_{k,N-K} \end{pmatrix}.$$

It is easy to see that the code generated by G_1 is equivalent to G, in the sense that the same codewords are generated, with permuted code bit positions.

The next step is to perform *hard-decision decoding* based on the k most reliable (MR) values of the permuted received sequence $\bar{z}$. Let $\bar{u}_a = (u_1, u_2, \cdots, u_K)$ denote the result. Then, the corresponding codeword in C_1 can be computed as

$$\bar{v} = \bar{u}_a G_1 = (v_0 \quad v_1 \quad \cdots \quad v_{K-1} \quad v_K \quad \cdots \quad v_N).$$

The estimated codeword $\bar{v}_{HD}$ in C can be obtained from $\bar{v}$, via the inverse mapping $\lambda_1^{-1}\lambda_2^{-1}$, as

$$\bar{v}_{HD} = \lambda_1^{-1}\left[\lambda_2^{-1}\left[\bar{v}\right]\right]. \tag{7.6}$$

Based on the hard-decision-decoded codeword $\bar{v}$, the algorithm proceeds to *re-process* it until either *practically optimum* or a predefined desired error performance is achieved.

Order-ℓ re-processing is defined as follows:

For $1 \leq i \leq \ell$, make all possible changes of i of the k MR bits of $\bar{v}$.

For each change, find the corresponding codeword $\bar{v}'$ in C_1 and map it onto a binary real sequence $\bar{x}'$, via the mapping $\{0,1\} \mapsto \{+1,-1\}$.

For each candidate codeword generated, compute the squared Euclidean distance, Equation (1.33) of Chapter 1, between the generated sequence $\bar{x}'$ and the reordered received sequence $\bar{z}$. After all the $\sum_{i=0}^{\ell}\binom{K}{i}$ codewords have been generated and their distances compared, the algorithm outputs the codeword $\bar{v}^*$ which is closest to $\bar{z}$.

From $\bar{v}^*$, using the inverse map (7.6), the most likely code sequence $\bar{v}^*_{\mathrm{ML}}$ is computed.

Re-processing can be done in a systematic manner to minimize the number of computations. In particular, as mentioned in Section 7.1, with binary transmission over an AWGN channel, there is no need to compute the Euclidean distance but rather the *correlation* between the generated codewords and the reordered received sequence. Thus the computation of the binary real sequence $\bar{x}'$ is not needed. Only additions of the permuted received values z_i, with sign changes given by the generated $\bar{v}^*$, are required.

7.6 Generalized minimum distance decoding

In 1966, Forney [For3] introduced generalized minimum distance (GMD) decoding. The basic idea was to extend the notion of an erasure, by dividing the received values into *reliability classes*. The decoding strategy is similar to the Chase algorithm Type-III, with the use of *erasure patterns*. The GMD decoder works by declaring an increasing number of erasures within the $d-1$ least reliable symbols, and testing a *sufficient condition* for the optimality of the decoded word, until the condition is satisfied or a maximum number of erasures have been considered.

Let C be a linear block (N, K, d) code. Assume that there is an *errors-and-erasures decoder* that is capable of decoding any combination of e errors and s erasures within the capability of the code, i.e., $2e + s \leq d - 1$. Such decoders were presented in Section 3.5.6 for BCH codes and in Section 4.3.2 for RS codes.

Let $\bar{r} = (r_1, r_2, \cdots, r_N)$ be the received word from the output of the channel, $r_i = (-1)^{c_i} + w_i$, where w_i is a zero-mean Gaussian random variable with variance $N_0/2$, $i = 1, 2, \cdots, N$. It is assumed that r has been normalized (clipped) so that its components lie in the range $[-1, +1]$. As before, the sign bits of the received values represent the hard-decision received word,

$$\bar{z} = (\, z_1 \quad z_2 \quad \cdots \quad z_N \,), \qquad z_j = \mathrm{sgn}(r_j),\ 1 \leq j \leq N.$$

As with Chase algorithms and the OSD algorithm, the *reliabilities* of the received channel values are sorted, producing a list of indexes I_j, $j = 1, 2, \cdots, N$, such that

$$|r_{I_1}| \leq |r_{I_2}| \leq \cdots \leq |r_{I_N}|.$$

In the first round of decoding, the hard-decision received word $\bar{z}$ is fed into an *errors-only* algebraic decoder. Let $\hat{v}$ denote the resulting estimated codeword. The correlation metric of $\hat{v}$ with respect to the received word $\bar{r}$

$$\nu = \sum_{j=1}^{N} (-1)^{\hat{v}_j} \times r_j, \tag{7.7}$$

is computed. If the following sufficient condition is satisfied

$$\nu > n - d, \tag{7.8}$$

then $\hat{v}$ is accepted as the most likely codeword and decoding stops.

Otherwise, a new round of decoding is performed. This is accomplished by setting $s = 2$ erasures, in positions I_1 and I_2, and decoding the resulting word with an *errors-and-erasures* decoder. The correlation metric between $\bar{r}$ and the estimated codeword $\hat{v}$ is computed, as in (7.7), and then the sufficient condition (7.8) tested.

This GMD decoding process continues, if necessary, every round increasing the number of erasures by two, $s = s + 2$, until the maximum number of erasures ($s_{\max} = d - 1$) in the least reliable positions are tried. If at the end of GMD decoding no codeword is found, the output can be either an indication of a decoding failure or the hard-decision decoded codeword $\hat{v}_0$ obtained with $s = 0$.

7.6.1 Sufficient conditions for optimality

The condition used in GMD decoding can be improved and applied to other decoding algorithms that output lists of codewords, such as Chase and OSD. These algorithms are instances of *list decoding* algorithms. The acceptance criteria (7.8) is too restrictive, resulting in many codewords rejected, possibly including the most likely (i.e., selected by true MLD) codeword. Improved sufficient conditions on the optimality of a codeword have been proposed. Without proofs, two such conditions are listed below. Before their description, some definitions are needed.

Let $\bar{x}$ represent a BPSK *modulated codeword*, $\bar{x} = m(\bar{v})$, where $\bar{v} \in C$ and $x_i = (-1)^{v_i}$, for $1 \leq i \leq N$. See also (7.2). Let $S_e = \{i : \text{sgn}(x_i) \neq \text{sgn}(r_i)\}$ be the set of error positions, $U = \{I_j, j = 1, 2, \cdots, d\}$ the set of least reliable positions, and $T = \{i : \text{sgn}(x_i) = \text{sgn}(r_i),\ i \in U\}$. Then the *extended distance* or *correlation discrepancy* between a codeword $\bar{v}$ and a received word $\bar{r}$ is defined as [TP],

$$d_e(\bar{v}, \bar{r}) = \sum_{i \in S_e} |y_i|. \tag{7.9}$$

Improved criteria for finding an optimum codeword are based on upper bounds on (7.9) and by increasing the cardinality of the sets of positions tested. Two improvements to Forney's conditions are:

- **Taipale-Pursley** [TP] There exists an optimal codeword $\bar{x}_{\text{opt}}$ such that

$$d_e(\bar{x}_{\text{opt}}, \bar{r}) < \sum_{i \in T} |r_i|. \tag{7.10}$$

- **Kasami et al.** [KKTFL] There exists an optimal codeword $\bar{x}_{\text{opt}}$ such that

$$d_e(\bar{x}_{\text{opt}}, \bar{r}) < \sum_{i \in T_K} |r_i|, \tag{7.11}$$

where $T_K = \{i : \text{sgn}(x_i) \neq \text{sgn}(r_i),\ i \in U\}$.

Good references to GMD decoding, its extensions, and combinations with Chase algorithms are [KNIH],[Kam],[TKK],[FL4] and [TLFL].

7.7 List decoding

List decoding was introduced by Elias and Wozencraft (see [Eli3]). Most recently, list decoding of polynomial codes has received considerable attention, mainly caused by the papers written by Sudan and colleagues [Sud, Gur] on decoding RS codes beyond their error correcting capabilities. The techniques used, referred to as *Sudan algorithms*, use interpolation and factorization of bi-variate polynomials over extension fields. Sudan algorithms can be considered extensions of the Welch-Berlekamp algorithm [Ber2]. These techniques have been applied to soft-decision decoding of RS codes in [KV].

7.8 Soft-output algorithms

The previous sections of this chapter have been devoted to decoding algorithms that output the most likely coded *sequence* or codeword (or list of codewords). However, since the appearance of the revolutionary paper on turbo codes in 1993 [BGT], there is a need for decoding algorithms that output not only the most likely codeword (or list of codewords), but also an estimate of the *bit reliabilities* for further processing. In the field of error correcting codes, soft-output algorithms were introduced as early as 1962, when Gallager [Gal] published his work on low-density parity-check (LDPC) codes[3], and later by Bahl et al. [BCJR]. In both cases, the algorithms perform a *forward-backward* recursion to compute the reliabilities of the code symbols. In the next section, basic soft-output decoding algorithms are described. Programs to simulate these decoding algorithms can be found on the ECC web site.

In the following sections, and for simplicity of exposition, it is assumed that a linear block code, constructed by terminating a binary memory-m rate-$1/n$ convolutional code, is employed for binary transmission over an AWGN channel. It is also assumed that the convolutional encoder starts at the all-zero state $S_0^{(0)}$ and, after N trellis stages, ends at the all-zero state $S_N^{(0)}$.

7.8.1 Soft-output Viterbi algorithm

In 1989, the Viterbi algorithm was modified to output bit reliability information [HH]. The *soft-output viterbi algorithm (SOVA)* computes the reliability, or soft-output, of the information bits as a *log-likelihood ratio* (LLR),

$$\Lambda(u_i) \triangleq \log\left(\frac{\Pr\{u_i = 1|\bar{r}\}}{\Pr\{u_i = 0|\bar{r}\}}\right), \tag{7.12}$$

[3] LDPC codes are covered in Chapter 8.

where $\bar{r}$ denotes the received sequence.

The operation of a SOVA decoder can be divided into two parts. In the first part, decoding proceeds as with the conventional VA, selecting the most likely coded sequence, $\hat{v}$, in correspondence with the path in the trellis with the maximum (correlation) metric, at stage n. (See Section 5.4.) In addition, the path metrics need to be stored at each decoding stage, and for each state. These metrics are needed in the last part of the algorithm, to compute the *soft outputs*. In the second part of SOVA decoding, the Viterbi algorithm transverses the trellis backwards, and computes metrics and paths, starting at $i = N$ and ending at $i = 0$. It should be noted that, in this stage of the SOVA algorithm, there is no need to store the surviving paths, but only the metrics for each trellis state. Finally for each trellis stage i, $1 \leq i \leq N$, the soft outputs are computed.

Let $M_{\max}$ denote the (correlation) metric of the most likely sequence $\hat{v}$ found by the Viterbi algorithm. The probability of the associated information sequence $\hat{u}$ given the received sequence, or *a-posteriori probability* (APP), is proportional to $M_{\max}$, since

$$\Pr\{\hat{u}|\bar{r}\} = \Pr\{\hat{v}|\bar{r}\} \sim e^{M_{\max}}. \tag{7.13}$$

Without loss of generality, the APP of information bit u_i can be written as

$$\Pr\{u_i = 1|\bar{r}\} \sim e^{M_i(1)},$$

where $M_i(1) \triangleq M_{\max}$. Let $M_i(0)$ denote the maximum metric of paths associated with the complement of information symbol u_i. Then it is easy to show that

$$\Lambda(u_i) \sim M_i(1) - M_i(0). \tag{7.14}$$

Therefore, at time i, the soft output can be obtained from the difference between the maximum metric of paths in the trellis with $\hat{u}_i = 1$ and the maximum metric of paths with $\hat{u}_i = 0$.

In the soft-output stage of the SOVA algorithm, at stage i, the most likely information symbol $u_i = a$, $a \in \{0, 1\}$, is determined and the corresponding maximum metric (found in the forward pass of the VA) set equal to $M_i(u_i)$. The path metric of the best competitor, $M_i(u_i \oplus 1)$, can be computed as [Vuc],

$$M_i(v_i \oplus 1) = \min_{k_1, k_2} \left\{ M_f(S_{i-1}^{(k_1)}) + BM_i^{(b_1)}(u_i \oplus 1) + M_b(S_i^{(k_2)}) \right\}, \tag{7.15}$$

where $k_1, k_2 \in \{0, 1, 2, \cdots, 2^m - 1\}$,

- $M_f(S_{i-1}^{(k_1)})$ is the path metric of the forward survivor at time $i-1$ and state $S^{(k_1)}$,
- $BM_i^{(b_1)}(u_i \oplus 1)$ is the branch metric at time i for the complement information associated with a transition from state $S^{(k_1)}$ to $S^{(k_2)}$, and
- $M_b(S_i^{(k_2)})$ is the backward survivor path metric at time i and state $S^{(k_2)}$.

Finally, the soft output is computed as

$$\Lambda(u_i) = M_i(1) - M_i(0). \tag{7.16}$$

Example 88 Let C be a zero-tail $(12, 4)$ code obtained from a memory-2 rate-1/2 convolutional code with generators $(7, 5)$. The basic structure of the trellis diagram of this

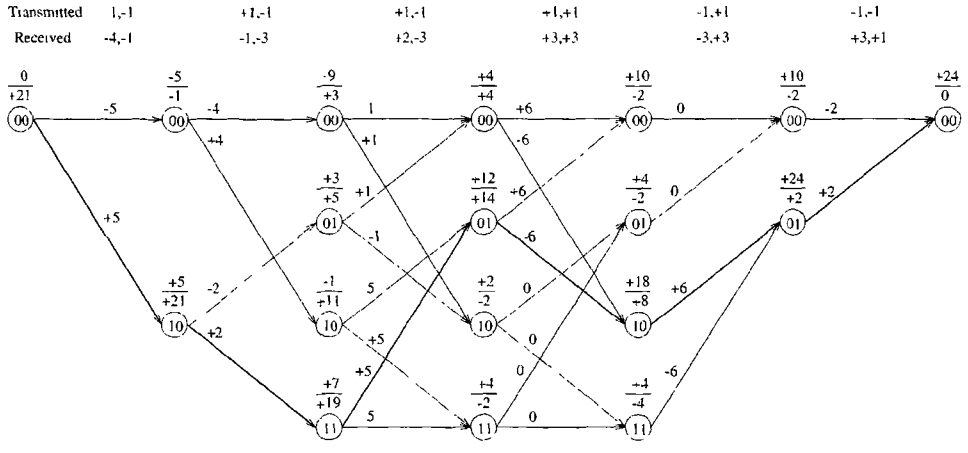


Figure 77 Trellis diagram used in SOVA decoding for Example 88.

code is the same as in Example 86. Suppose that the information sequence (including the tail bits) is $\bar{u} = (110100)$, and that the received sequence, after binary transmission over an AWGN channel, is

$$\bar{r} = (-4, -1, -1, -3, +2, -3, +3, +3, -3, +3, -3, +1).$$

Figure 77 shows the trellis diagram for this code. For $i = 0, 1, \cdots, 6$, each state has a label on top of it of the form

$$\frac{M_f(S_i^{(m)})}{M_b(S_i^{(m)})}.$$

The branches are labeled with the branch metrics BM_i. The soft outputs $\Lambda(u_i)$, for $i = 1, 2, \cdots, 6$ are $-34, -16, +22, -16, +24, +24$, respectively.

Implementation issues

In a SOVA decoder, the Viterbi algorithm needs to be executed twice. The forward processing is just as in conventional VD, with the exception that path metrics at each decoding stage need to be stored. The backward processing uses the VA, but does not need to store the surviving paths, only the metrics at each decoding state. Note that both backward processing and soft-decision computation can be done simultaneously. The backward processing stage does not need to store path survivors. In addition, the soft outputs need to be computed, after both the forward and backward recursions finish. Particular attention should be paid to the normalization of metrics at each decoding stage, in both directions. Other implementation issues are the same as in a Viterbi decoder, discussed in Sections 5.4.3 and 5.5.1.

The SOVA decoder can also be implemented as a *sliding window decoder*, like the conventional Viterbi decoder. By increasing the computation time, the decoder operates continuously, not on a block-by-block basis, without forcing the state of the encoder to return to the all-zero state periodically. The idea is the same as that used in the VD with traceback memory, as discussed in Section 5.4.3, where forward recursion, traceback and

backward recursion, and soft-output computations are implemented in several memory blocks (see also [Vit3]).

7.8.2 *Maximum-a-posteriori (MAP) algorithm*

The BCJR algorithm [BCJR] is an optimal symbol-by-symbol MAP decoding method for linear block codes that minimizes the probability of a symbol error. The goal of this MAP decoder is to examine the received sequence $\bar{r}$ and to compute the a-posteriori probabilities of the input information bits, as in (7.12). The MAP algorithm is described next, following closely the arguments in [BCJR].

The state transitions (or branches) in the trellis have probabilities

$$\Pr\left\{S_i^{(m)}|S_{i-1}^{(m')}\right\}, \tag{7.17}$$

and for the output symbols $\bar{v}_i$,

$$q_i(x_i|m',m) \triangleq \Pr\left\{x_i = x|S_{i-1}^{(m')}, S_i^{(m)}\right\}, \tag{7.18}$$

where $x = \pm 1$, and $x_i = m(v_i) = (-1)^{v_i}$, $0 < i \leq N$.

The sequence $\bar{x}$ is transmitted over an AWGN channel and received as a sequence $\bar{r}$, with transition probabilities

$$\Pr\{\bar{r}|\bar{x}\} = \prod_{i=1}^{N} p(\bar{r}_i|\bar{x}_i) = \prod_{i=1}^{N}\prod_{j=0}^{n-1} p(r_{i,j}|x_{i,j}), \tag{7.19}$$

where $p(r_{i,j}|x_{i,j})$ is given by (7.1).

Let $B_i^{(j)}$ be the set of branches connecting state $S_{i-1}^{(m')}$ to state $S_i^{(m)}$ such that the associated information bit $u_i = j$, with $j \in \{0,1\}$. Then

$$\Pr\{u_i = j|\bar{r}\} = \sum_{(m',m)\in B_i^{(j)}} \Pr\left\{S_{i-1}^{(m')}, S_i^{(m)}, \bar{r}\right\} \triangleq \sum_{(m',m)\in B_i^{(j)}} \sigma_i(m',m). \tag{7.20}$$

The value of $\sigma_i(m',m)$ in (7.20) is equal to

$$\sigma_i(m',m) = \alpha_{i-1}(m') \cdot \gamma_i^{(j)}(m',m) \cdot \beta_i(m), \tag{7.21}$$

where the joint probability $\alpha_i(m) \triangleq \Pr\left\{S_i^{(m)}, \bar{r}_p\right\}$ is given recursively by

$$\alpha_i(m) = \sum_{m'} \alpha_{i-1}(m') \cdot \sum_{j=0}^{1} \gamma_i^{(j)}(m',m), \tag{7.22}$$

and is referred to as the *forward metric*.

The conditional probability $\gamma_i^{(j)}(m',m) \triangleq \Pr\left\{S_i^{(m)}, \bar{r}|S_{i-1}^{(m')}\right\}$ is given by

$$\gamma_i^{(j)}(m',m) = \sum_x p_i(m|m')\Pr\left\{x_i = x|S_{i-1}^{(m')}, S_i^{(m)}\right\} \cdot \Pr\{r_i|x\} \tag{7.23}$$

where $p_i(m|m') = \Pr\left\{S_i^{(m)}|S_{i-1}^{(m')}\right\}$, which for the AWGN channel can be put in the form

$$\gamma_i^{(j)}(m', m) = \Pr\{u_i = j\} \cdot \delta_{ij}(m, m') \cdot \exp\left(-\frac{1}{N_0}\sum_{q=0}^{n-1}(r_{i,q} - x_{i,q})^2\right), \tag{7.24}$$

where $\delta_{ij}(m, m') = 1$ if $\{m', m\} \in B_i^{(j)}$, and $\delta_{ij}(m, m') = 0$ otherwise. $\gamma_i^{(j)}(m', m)$ is referred to as the *branch metric*.

The conditional probability $\beta_i(m) \triangleq \Pr\left\{\bar{r}_f|S_i^{(m)}\right\}$ is given by

$$\beta_i(m) = \sum_{m'} \beta_{i+1}(m') \cdot \sum_{j=0}^{1} \gamma_i^{(j)}(m', m), \tag{7.25}$$

and referred to as the *backward metric*.

Combining (7.25), (7.24), (7.22) , (7.21) and (7.12), the soft output (LLR) of information bit u_i is given by

$$\Lambda(u_i) = \log\left(\frac{\Pr\{u_i = 1|r\}}{\Pr\{u_i = 0|\bar{r}\}}\right) = \log\left(\frac{\sum_m \sum_{m'} \alpha_{i-1}(m')\gamma_i^{(1)}(m', m)\beta_i(m)}{\sum_m \sum_{m'} \alpha_{i-1}(m')\gamma_i^{(0)}(m', m)\beta_i(m)}\right), \tag{7.26}$$

with the hard-decision output is given by $\hat{u}_i = \text{sgn}(\Lambda(u_i))$ and the reliability of the bit u_i is $|\Lambda(u_i)|$. The above equations can be interpreted as follows. A bi-directional Viterbi-like algorithm can be applied, just as in SOVA decoding (previous section). In the forward recursion, given the probability of a state transition at time i, the joint probability of the received sequence up to time i and the state at time i is evaluated. In the backward recursion, the probability of the received sequence from time $i+1$ to time N, given the state at time i is computed. Then the soft output depends on the joint probability of the state transition and the received symbol at time i.

The MAP algorithm can be summarized as follows.

- **Initialization**

  For $m = 0, 1, \cdots, 2^m - 1$,

  $$\alpha_0(0) = 1, \quad \alpha_0(m) = 0, m \neq 0,$$

  For $m' = 0, 1, \cdots, 2^m - 1$,

  $$\beta_N(0) = 1, \quad \beta_N(m') = 0, m' \neq 0,$$

- **Forward recursion**

  For $i = 1, 2, \cdots, N$,

  1. For $j = 0, 1$, compute and store the branch metrics $\gamma_i^{(j)}(m', m)$ as in (7.24).
  2. For $m = 0, 1, ..., 2^m - 1$, compute and store the forward metrics $\alpha_i(m)$ as in (7.22).

- **Backward recursion**

  For $i = N-1, N-2, \cdots, 0$,

  1. Compute the backward metrics $\beta_i(m)$ as in (7.25), using the branch metric computed in the forward recursion.
  2. Compute the log-likelihood ratio $\Lambda(u_i)$ as in (7.26).

Implementation issues

The implementation of a MAP decoder is similar to that of a SOVA decoder, as both decoders perform forward and backward recursions. All the issues mentioned in the previous section apply to a MAP decoder. In addition, note that branch metrics depend on the noise power density N_0, which should be estimated to keep optimality. To avoid numerical instabilities, the probabilities $\alpha_i(m)$ and $\beta_i(m)$ need to be scaled at every decoding stage, such that $\sum_m \alpha_i(m) = \sum_m \beta_i(m) = 1$.

7.8.3 Log-MAP algorithm

To reduce the computational complexity of the MAP algorithm, the logarithms of the metrics may be used. This results in the so-called *log-MAP* algorithm.

From (7.22) and (7.25) [RVH],

$$\begin{aligned} \log \alpha_i(m) &= \log\left(\sum_{m'}\sum_{j=0}^{1} \exp\left(\log \alpha_{i-1}(m') + \log \gamma_i^{(j)}(m', m)\right)\right), \\ \log \beta_i(m) &= \log\left(\sum_{m'}\sum_{j=0}^{1} \exp\left(\log \beta_{i+1}(m') + \log \gamma_i^{(j)}(m', m)\right)\right). \end{aligned} \tag{7.27}$$

Taking the logarithm of $\gamma_i^{(j)}(m', m)$ in (7.24),

$$\log \gamma_i^{(j)}(m', m) = \delta_{ij}(m.m')\left\{\log \Pr\{u_i = j\} - \frac{1}{N_0}\sum_{q=0}^{n-1}(r_{i,q} - x_{i,q})^2\right\} \tag{7.28}$$

By defining $\bar{\alpha}_i(m) = \log \alpha_i(m)$, $\bar{\beta}_i(m) = \log \beta_i(m)$ and $\bar{\gamma}_i^{(j)}(m', m) = \log \gamma_i^{(j)}(m', m)$, (7.26) can be written as

$$\Lambda(u_i) = \log\left(\frac{\sum_m \sum_{m'} \exp\left(\bar{\alpha}_{i-1}(m') + \bar{\gamma}_i^{(0)}(m', m) + \bar{\beta}_i(m)\right)}{\sum_m \sum_{m'} \exp\left(\bar{\alpha}_{i-1}(m') + \bar{\gamma}_i^{(1)}(m', m) + \bar{\beta}_i(m)\right)}\right), \tag{7.29}$$

and an algorithm that works in the log-domain is obtained.

The following expression, known as the Jacobian logarithm [RVH], is used to avoid the sum of exponential terms,

$$\log\left(e^{\delta_1} + e^{\delta_2}\right) = \max(\delta_1, \delta_2) + \log\left(1 + e^{-|\delta_1 - \delta_2|}\right), \tag{7.30}$$

The function $\log\left(1 + e^{-|\delta_1 - \delta_2|}\right)$ can be stored in a small look-up table (LUT), as only a few values (eight reported in [RVH]) are required to achieve practically the same performance as the MAP algorithm. Therefore, instead of several calls to slow (or hardware expensive) $\exp(x)$ functions, simple LUT accesses give practically the same result.

7.8.4 *Max-Log-MAP algorithm*

A more computationally efficient, albeit suboptimal derivative of the MAP algorithm is the *Max-Log-MAP algorithm*. It is obtained as before, by taking the logarithms of the MAP metrics, and using the approximation [RVH]

$$\log\left(e^{\delta_1} + e^{\delta_2}\right) \approx \max\left(\delta_1, \delta_2\right), \tag{7.31}$$

which is equal to the first term on the right-hand side of (7.30). As a result, the log-likelihood ratio of information bit u_i is given by

$$\begin{aligned}\Lambda(u_i) \approx & \max_{m',m}\left\{\bar{\alpha}_{i-1}(m') + \bar{\gamma}_i^{(0)}(m',m) + \bar{\beta}_i(m)\right\} \\ & - \max_{m',m}\left\{\bar{\alpha}_{i-1}(m') + \bar{\gamma}_i^{(1)}(m',m) + \bar{\beta}_i(m)\right\}.\end{aligned} \tag{7.32}$$

The forward and backward computations can now be expressed as

$$\begin{aligned}\bar{\alpha}_i(m) &= \max_{m'} \max_{j\in\{0,1\}}\left\{\bar{\alpha}_{i-1}(m') + \bar{\gamma}_i^{(j)}(m',m)\right\}, \\ \bar{\beta}_i(m) &= \max_{m'} \max_{j\in\{0,1\}}\left\{\bar{\beta}_{i+1}(m') + \bar{\gamma}_i^{(j)}(m',m)\right\}.\end{aligned} \tag{7.33}$$

For binary codes based on rate-$1/n$ convolutional encoders, in terms of decoding complexity (measured in number of additions and multiplications), the SOVA algorithm requires the least amount, about half of that of the max-log-MAP algorithm. The log-MAP algorithm is approximately twice more complex compared to the max-log-MAP algorithm. In terms of performance, it has been shown [FBLH] that the max-log-MAP algorithm is equivalent to a modified SOVA algorithm. The log-MAP and MAP algorithms have the same best error performance.

7.8.5 *Soft-output OSD algorithm*

The OSD algorithm of Section 7.5 can be modified to output the symbol reliabilities [FL5]. This modification is referred to as the *soft-output OSD*, or SO-OSD. The SO-OSD algorithm is a two-stage order-i re-processing. The first stage is the same as conventional OSD, determining the most likely codeword $\bar{v}_{\mathrm{ML}}$ up to order-i re-processing. To describe the second stage, the following definitions are required.

For each most reliable position j, $1 \le j \le K$, define the codeword $\bar{v}_{\mathrm{ML}}(j)$ is obtained by complementing position j in $\bar{v}_{\mathrm{ML}}$,

$$\bar{v}_{\mathrm{ML}}(j) = \bar{v}_{\mathrm{ML}} \oplus \bar{e}(j),$$

where $\bar{e}(j)$ is the set of all length-K vectors of Hamming weight one. The vector $\bar{e}(j)$ is the *coset representative* of the partition of code C_1 (equivalent to the original code after reordering, as in OSD, see Section 7.5) into two sets of codewords, having the j-th position equal to zero or one in codewords of C. These sets, after removing position j, become punctured subcodes of C_1, and are denoted $C(0)$ and $C(1)$. Let $C(j) = C(0)$.

The SO-OSD algorithm consists of the following steps:

1. Determine the most likely codeword $\bar{v}_{\mathrm{ML}}$ covered by order-i re-processing.

2. For each subcode $C(j)$, $1 \leq j \leq K$,
   (a) Initialize all soft output values of the least reliable positions (LRP) based on $\bar{v}_{\mathrm{ML}}$ and $\bar{v}_{\mathrm{ML}}(j)$. That is, compute

$$L_j = r_j + \sum_{\substack{\ell \neq j \\ v_\ell(0) \neq v_\ell(1)}} m(v_\ell(1)) r_\ell, \tag{7.34}$$

   where $m(v_i) = (-1)^{v_i}$.
   (b) Determine the most likely codeword $\bar{v}(j) \in C(j)$ covered by order-i reprocessing.
   (c) Evaluate L_j (7.34) based on $\bar{v}_{\mathrm{ML}}$ and $\bar{v}(j)$.
   (d) Update soft output values of LRP in the positions of $\bar{v}_{\mathrm{ML}} \oplus \bar{v}(j)$ with L_j.
3. For $K + 1 \leq j \leq N$, choose the smallest output value associated with each LRP j.

The performance of SO-OSD is the same as max-log-MAP decoding for many binary linear block codes of length up to 128 and high-rate codes [FL5]. In addition, scaling down the extrinsic values (7.34) improves the performance by a few tenths of a dB.

8

Iteratively decodable codes

Iterative decoding may be defined as a technique employing a soft-output decoding algorithm that is iterated several times to improve the error performance of a coding scheme, with the aim of approaching true maximum-likelihood decoding (MLD), with less complexity. When the underlying error correcting code is well designed, increasing the number of iterations results in an improvement of the error performance.

Iterative decoding techniques date back to 1954 with the work of Elias [Eli1] on *iterated codes*. Later, in the 1960s, Gallager [Gal] and Massey [Mas1] made important contributions. Iterative decoding was then referred to as *probabilistic decoding*. The main concept was then, as it is today, to maximize the *a-posteriori probability* of a symbol being sent given a noisy version of the coded sequence.

In this chapter, code constructions that can be iteratively decoded are presented. Elsewhere in the literature, these coding schemes have been named *turbo-like* codes. In terms of the application of iterative decoding algorithms, and for the purpose of this chapter, ECC schemes can be broadly classified into two classes:

Product codes

Examples of codes in this class include *parallel concatenated codes*, or *turbo codes*, and *serially concatenated codes*. Members of this class are also block product codes, presented in Section 6.2.3.

Low-density parity-check (LDPC) codes

These are linear codes with the property that their parity-check matrix has a small ratio of number of nonzero elements to the total number of elements.

In both of the above classes of codes, the component codes can be either convolutional or block codes, with systematic or nonsystematic encoding, or any combination thereof. To provide a motivation for the study of iterative decoding techniques, the fundamental structure of turbo codes is discussed next.

Turbo codes

Over the past eight years, there has been an enormous amount of research effort dedicated to the analysis of iterative decoding algorithms and the construction of *iteratively decodable codes* or "turbo-like codes" that approach the *Shannon limit*[1]. Turbo codes were introduced in

[1] The Shannon limit refers to the capacity of a discrete-input continuous output channel.

1993 [BGT]. The results reported there sent a shock wave throughout the research community.

With binary transmission over an AWGN channel, a rate-1/2 coding scheme, based on a combination of two rate-1/2 16-state RSC codes, connected with an interleaver of size 256×256, achieved BER rates of 10^{-5} at an signal-to-noise (SNR) ratio per bit, or E_b/N_0, of 0.7 dB, closer than ever before to the Shannon limit (0.2 dB). See Figure 78.

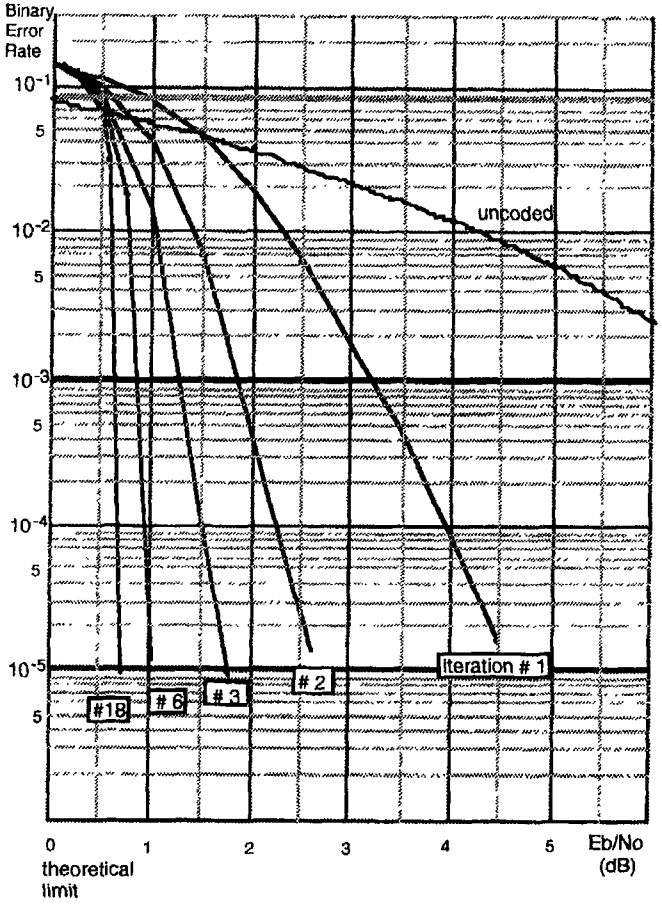


Figure 78 Performance of a rate-1/2 parallel concatenated (turbo) code with memory-4 rate-1/2 RSC codes, $\bar{g}_0 = 21$ and $\bar{g}_1 = 37$. Block interleaver size $256 \times 256 = 65536$. From [BG2]. Reproduced by permission of IEEE. (©1996 IEEE)

The basic elements in the turbo coding scheme proposed in [BGT] are the following:

Coding structure

A *product code* (referred to in the original paper [BGT] as *parallel concatenated code*) structure, with constituent *recursive systematic* convolutional encoders.

Reliability-based

Employ *soft-input soft-output (SISO)* maximum-a-posteriori (MAP) decoders for the component codes, in order to generate *log-likelihood ratios*.

Iterative decoding

The use of *feedback* of part of the symbol reliabilities, in the form of *extrinsic information*, from an outer (column) decoder to an inner (row) decoder, and vice versa.

Random permutation

A long *random interleaver* applied between the two encoders. Its main function is to ensure that, at each iteration, the component MAP decoders get independent estimates on the information symbols.

These four elements have been extensively studied over the years. [Vuc, ISTC1, ISTC2, JSAC1, JSAC2, HeW] are a good sample of the research efforts.

8.1 Iterative decoding

The purpose of this section is to introduce basic concepts behind iterative decoding. Consider the a-posteriori log-likelihood ratio (LLR)[2] of an information symbol, Equation (7.12) on page 134, repeated here for convenience.

$$\Lambda(u_i) \triangleq \log\left(\frac{\Pr\{u_i = 1|\bar{r}\}}{\Pr\{u_i = 0|\bar{r}\}}\right). \tag{8.1}$$

Let $C_i(0)$ and $C_i(1)$ denote the set of modulated coded sequences $\bar{x} = m(\bar{v}) \in m(C)$ such that the i-th position in $\bar{v} \in C$ is equal to zero and one, respectively. Recall that modulation was a one-to-one and onto mapping from $v \in \{0,1\}$ to $x \in \{+1,-1\}$, as in Equation (7.2). Then the symbol LLR (8.1) can be written as,

$$\begin{aligned}\Lambda(u_i) &= \log\left(\frac{\displaystyle\sum_{\bar{x}\in C_i(1)}\prod_{\ell=1}^{N} p(r_\ell, x_\ell)}{\displaystyle\sum_{\bar{x}\in C_i(0)}\prod_{\ell=1}^{N} p(r_\ell, x_\ell)}\right) \\ &= \log\left(\frac{p(r_i|x_i=-1)\Pr\{x_i=-1\}\displaystyle\sum_{\bar{x}\in C_i(1)}\prod_{\substack{\ell=1\\ \ell\neq i}}^{N} p(r_\ell, x_\ell)}{p(r_i|x_i=+1)\Pr\{x_i=+1\}\displaystyle\sum_{\bar{x}\in C_i(0)}\prod_{\substack{\ell=1\\ \ell\neq i}}^{N} p(r_\ell, x_\ell)}\right),\end{aligned} \tag{8.2}$$

from which it follows that the LLR of an information symbol can be expressed as

[2] The LLR is also referred to as "L-value" by some authors.

$$\Lambda(u_i) = \Lambda_{ci} + \Lambda_a(u_i) + \Lambda_{i,e}(C), \tag{8.3}$$

where Λ_{ci} is known as the *channel LLR* and given by

$$\begin{aligned}\Lambda_{ci} &= \log\left(\frac{p(r_i|x_i=-1)}{p(r_i|x_i=+1)}\right)\\ &= \begin{cases}(-1)^{r_i}\log\left(\frac{1-p}{p}\right), & \text{for a BSC channel with parameter } p;\\ \frac{4}{N_0}r_i, & \text{for an AWGN channel } (\sigma_n^2 = N_0/2); \text{ and}\\ \frac{4}{N_0}a_i r_i, & \text{for a flat Rayleigh fading channel,}\end{cases}\end{aligned} \tag{8.4}$$

the quantity

$$\Lambda_a(u_i) = \log\left(\frac{\Pr\{x_i=-1\}}{\Pr\{x_i=+1\}}\right) = \log\left(\frac{\Pr\{u_i=1\}}{\Pr\{u_i=0\}}\right), \tag{8.5}$$

is the *a-priori LLR* of the information symbol, and

$$\Lambda_{i,e}(C) = \log\left(\frac{\sum_{\bar{x}\in C_i(1)}\prod_{\substack{\ell=1\\ \ell\neq i}}^{N} p(r_\ell, x_\ell)}{\sum_{\bar{x}\in C_i(0)}\prod_{\substack{\ell=1\\ \ell\neq i}}^{N} p(r_\ell, x_\ell)}\right), \tag{8.6}$$

is the *extrinsic LLR*, which is specified by the constraints imposed by the code on the *other information symbols*.

Assume that binary transmission is performed over an AWGN channel and that binary rate-$1/n$ RSC encoders are used as components. In an iterative decoding procedure, the extrinsic information provided by $\Lambda_{i,e}(C)$ can be fed back to the decoder as a-priori probability for a second round of decoding. In terms of forward-backward decoding using a trellis (see Section 7.8.2), the extrinsic LLR can be written as

$$\Lambda_{i,e}(C) = \log\left(\frac{\sum_{(m,m')\in B_i(1)}\alpha_{i-1}(m')\xi_i(m',m)\beta_i(m)}{\sum_{(m,m')\in B_i(0)}\alpha_{i-1}(m')\xi_i(m',m)\beta_i(m)}\right), \tag{8.7}$$

where $\alpha_i(m)$ and $\beta_i(m)$ are given by (7.22) and (7.25), respectively.

A *modified branch metric* is needed to compute the extrinsic LLR,

$$\xi_i(m',m) = \delta_{ij}(m,m')\exp\left(\frac{E}{N_0}\sum_{q=1}^{n-1} r_{i,q}x_{i,q}\right), \tag{8.8}$$

where, as before,

$$\delta_{ij}(m,m') = \begin{cases}1, & \text{if } (m',m)\in B_i^{(j)};\\ 0, & \text{otherwise.}\end{cases}$$

This extrinsic LLR, for an information position i, does not contain any variable directly related to u_i, for $i = 1, 2, \cdots, N$. Because it is assumed that encoding is systematic, notice that the branch labels in the trellis are of the form $(u_i, v_{i,1}, \cdots, v_{i,n-1})$, and therefore the sum in the modified branch metric (8.8) starts at $q = 1$.[3]

In the more general case of a two-dimensional product coding scheme, the first (e.g., row) decoder produces $\Lambda_{i,e}^{(1)}(C)$ which is given to the second (e.g., column) decoder as a-priori probability $\Lambda_a^{(2)}(u_i)$ to be used in the computation of the LLR of information symbol u_i. In other words, the extrinsic information provides a soft output that involves only soft inputs (reliabilities) that are not directly related to the information symbol u_i. Iterative decoding of product codes is the topic of the next section.

8.2 Product codes

In this section, coding schemes based on products of codes with interleaving are presented. These schemes allow the use of simple decoders for the component codes. With iterative decoding, symbol-by-symbol MAP decoders of the component codes can exchange extrinsic LLR values, generally improving the reliability of the estimated information bits, with increasing number of iterations.

8.2.1 Parallel concatenation: turbo codes

Figure 79 shows the block diagram of an encoder of a *parallel concatenated code*, better known as a *turbo code*. Two encoders are used, each delivering as output only redundant symbols. For instance, if rate-1/2 RSC codes are used, the encoders represent the term $\bar{g}_1(D)/\bar{g}_0(D)$ in the polynomial generator matrix $G(D)$. The inputs to the encoders are $\bar{u}$ and $\Pi\bar{u}$, where Π denotes a *permutation matrix*[4] associated with the interleaver. The output corresponding to an input symbol u_i is $(\, u_i \quad v_{P1,i} \quad v_{P2,i}\,)$, with $v_{P1,i}$ and $v_{P2,i}$ representing the redundant symbols, for $i = 1, 2, \cdots, N$, where N is the block length.

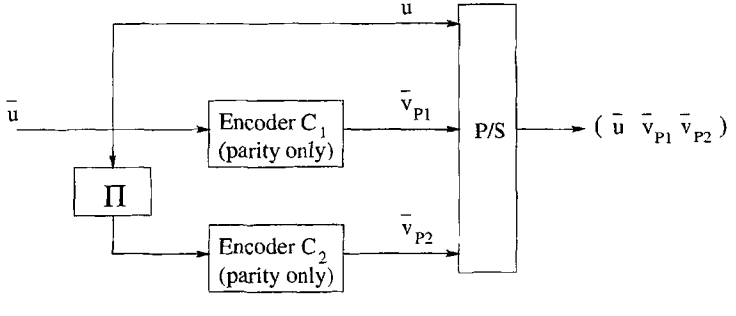


Figure 79 Encoder structure of a parallel concatenated (turbo) code.

[3] Compare this with the expression of the branch metric $\gamma_i^{(j)}(m', m)$ in (7.24).

[4] A permutation matrix is a binary matrix with only one nonzero entry per row and per column, if $\pi_{i,j} = 1$ then a symbol in position i at the input is sent to position j at the output.

The original turbo code scheme used RSC encoders. However, to examine this coding scheme, assume that the component codes are two binary *systematic* linear block (n_i, k_i) codes, $i = 1, 2$. The rate of the overall code is (see Figure 80)

$$R = \frac{K}{N} = \frac{k_1 k_2}{n_1 n_2 - (n_1 - k_1)(n_2 - k_2)} = \frac{R_1 R_2}{R_1 + R_2 + R_1 R_2}, \tag{8.9}$$

where $R_i \triangleq k_i/n_i$, $i = 1, 2$. Let $G_i = (I_i | P_i)$ denote the generator matrix of code C_i, $i = 1, 2$. Then the generator matrix of the parallel concatenated code $C_{\rm PC}$ can be put in the following form,

$$\begin{aligned} G_{\rm PC} &= \left(I_{k_1 k_2} \quad \begin{pmatrix} P_1 & & \\ & P_1 & \\ & & \ddots \\ & & & P_1 \end{pmatrix} \quad \Pi \begin{pmatrix} P_2 & & \\ & P_2 & \\ & & \ddots \\ & & & P_2 \end{pmatrix} \right) \\ &= (I_{k_1 k_2} \mid P_1' \mid P_2'), \end{aligned} \tag{8.10}$$

where Π is the permutation matrix associated with the interleaver, and P_i is the parity submatrix of code C_i, $i = 1, 2$. The number of times that P_1 appears in the middle part P_1' of $G_{\rm PC}$ is k_2, while the number of times that P_2 appears in the leftmost portion P_2' of $G_{\rm PC}$ is k_1. All other entries in P_1' and P_2' are zero. It follows that codewords of $C_{\rm PC}$ are of the form $(\bar{u} \mid \bar{u}P_1' \mid \bar{u}P_2')$.

Example 89 Let C_1 be a binary repetition $(2, 1, 2)$ code and C_2 be an binary SPC $(3, 2, 2)$ code. Then $k_1 k_2 = 2$. There is only one possible permutation matrix Π in this case. The parity submatrices and permutation matrix are

$$P_1 = (1), \quad P_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \Pi = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

respectively. From (8.10) it follows that the generator matrix of the parallel concatenated $(5, 2)$ code $C_{\rm PC}$ is

$$\begin{aligned} G_{\rm PC} &= \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} (1) & 0 \\ 0 & (1) \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) \\ &= \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} (1) & 0 \\ 0 & (1) \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) \\ &= \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}. \end{aligned}$$

Note that the minimum distance is $d_{\rm PC} = 3$, which is less than that of the product of C_1 and C_2.

One way to interpret a parallel concatenated (turbo) code, from the point of view of linear block codes, is as a *punctured product code*, in which the redundant symbols corresponding with the checks-on-checks are deleted. This interpretation is shown in Figure 80. The only essential difference between a turbo code and a block product code, is that the interleaver is not simply a row-by-row column-by-column interleaver, but one that introduces sufficient

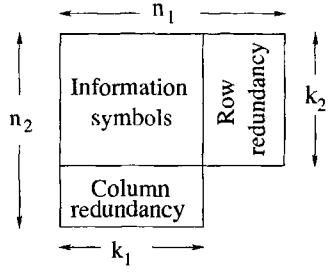


Figure 80 A parallel concatenated code interpreted as a *punctured product code.*

disorder in the information sequence, so that iterative decoding works. When interpreted as a linear block code, a turbo code can be further interpreted as a *generalized concatenated code*[5], after replacing the puncturing operation with an equivalent multiplication by a binary matrix and a direct-sum. Let M_1 be a $1 \times N$ matrix with $n_1 k_2$ successive ones followed by $N - n_1 k_2$ zeros. Let M_2 be another $1 \times N$ matrix with $N - (n_2 - k_2)k_1$ zeros followed by $(n_2 - k_2)k_1$ ones. Then a turbo encoder can be visualized as an encoder of a GC code, as shown in Figure 81.

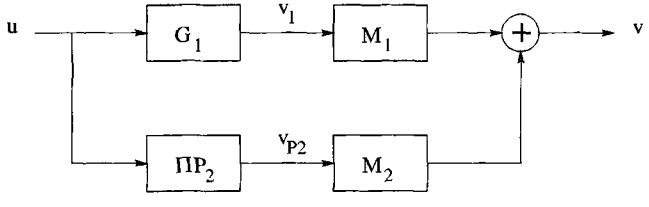


Figure 81 A turbo encoder as an encoder of a *generalized concatenated code.*

Iterative decoding of parallel concatenated codes

The basic structure of an iterative decoder for a parallel concatenated coding scheme with two component codes is shown in Figure 82. Each iteration consists of two phases, one decoding phase per component decoder, as follows.

First phase

In the first decoding iteration, first phase, the soft-in soft-out (SISO) decoder for the first component code computes the a-posteriori LLR (8.7) assuming equally likely symbols, i.e., $\Lambda_a(\bar{u}) = 0$. This decoder computes the extrinsic information for each information symbol, $\Lambda_{e1}(\bar{u})$, based on the part of the received sequence that corresponds to the parity symbols, $\bar{r}_{P1}$, and sends the result to the second SISO decoder.

---

[5] See Section 6.2.5.

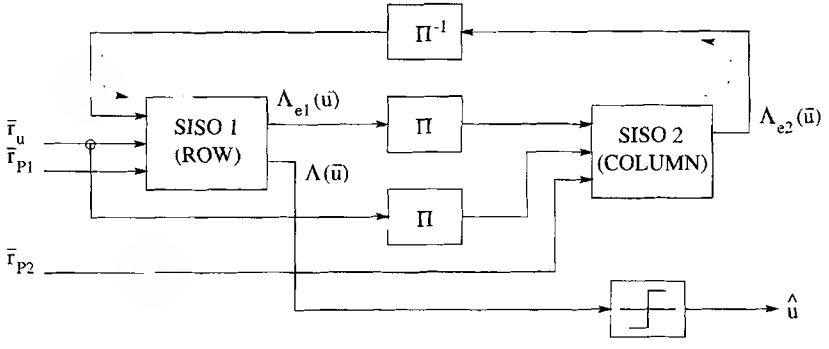


Figure 82 Block diagram of an iterative decoder for a parallel concatenated code.

Second phase

In the second phase of first decoding iteration, the permuted (or interleaved) extrinsic information from the first decoder is used as a-priori LLR, $\Lambda_a(\bar{u}) = \Pi\Lambda_{e1}(\bar{u})$. Extrinsic information $\Lambda_{e2}(\bar{u})$ is computed, based on the part of the received sequence that corresponds to the parity symbols of the second component code, $\bar{r}_{P2}$, thus terminating the first decoding iteration. At this point, a decision can be made on an information symbol, based on its a-posteriori LLR $\Lambda(u)$.

In subsequent iterations, the first decoder uses the deinterleaved extrinsic information from the second decoder, $\Pi^{-1}\Lambda_{e2}(\bar{u})$, as a-priori LLR for the computation of the soft output (the a-posteriori LLR), $\Lambda(u)$. This procedure can be repeated until either a stopping criterion is met [HOP, SLF], or a maximum number of iterations is performed. Note that making decisions on the information symbols after the first (row) decoder saves one deinterleaver.

As in Section 7.8.4, the iterative MAP decoding algorithm described above can be simplified by using the approximation of the "log" by the "max" as in (7.31). Also, there is an iterative version of the SOVA algorithm presented in Section 7.8.1, that essentially computes the a-posteriori LLR $\Lambda(\bar{u})$ directly and then, subtracting the channel LLR $\bar{\Lambda}_c$ and the extrinsic LLR $\Lambda_{ej}(\bar{u})$ from the other decoder, produces the extrinsic LLR [FV].

Example 90 Consider the turbo $(5, 2, 3)$ code of Example 89. A codeword is arranged in a 3×2 array with the lower right corner deleted. Suppose that the following codeword $\bar{v}$

$u_1 = 0$	$v_{11} = 0$
$u_2 = 1$	$v_{12} = 1$
$v_{21} = 1$	X

(where "X" denotes that the position is deleted or ignored), is (BPSK) modulated and transmitted over an AWGN channel and received as (this is the channel LLR $\bar{\Lambda}_c$):

-0.5	+2.5
-3.5	+1.0
+0.5	X

For the computations below, we use the fact that, for two binary variables x_1 and x_2, the LLR of their sum $x_1 + x_2$ is [HOP]

$$\Lambda(x_1 + x_2) = \log\left[\frac{\exp(\Lambda(x_1) + \Lambda(x_2))}{(1 + \exp(\Lambda(x_1))) + (1 + \exp(\Lambda(x_2)))}\right]. \tag{8.11}$$

First iteration

In the first decoding phase, the row decoder for the repetition code, computes

$$\begin{aligned}\Lambda_{e1}(u_1) &= \Lambda(r_{11}) = +2.5,\\ \Lambda_{e1}(u_2) &= \Lambda(r_{12}) = +1.0.\end{aligned}$$

In the second decoding phase, the extrinsic information from the row decoder is used as a-priori LLR for the column decoder for the parity-check code. The updated LLRs are

+2.0	+2.5
-2.5	+1.0
+0.5	X

In the second phase of the first iteration, the column decoder computes the extrinsic LLRs,

$$\Lambda_{e2}(u_1) = \Lambda(u_2 + r_{21}) = \log\left[\frac{1+e^{-2}}{e^{-2.5}+e^{+0.5}}\right] = -0.42,$$

$$\Lambda_{e2}(u_2) = \Lambda(u_1 + r_{21}) = \log\left[\frac{1+e^{+2.5}}{e^{+2}+e^{+0.5}}\right] = +0.38.$$

At the end of the first decoding iteration, the a-posteriori LLR are

$$\begin{aligned}\Lambda(u_1) &= \Lambda_{c1} + \Lambda_{e1}(u_1) + \Lambda_{e2}(u_1) = +1.58\\ \Lambda(u_2) &= \Lambda_{c2} + \Lambda_{e1}(u_2) + \Lambda_{e2}(u_2) = -2.13\end{aligned}$$

The iterative decoder could make a decision at this point, based on the sign of the a-posteriori LLR, and deliver the (correct) estimates $\hat{u}_1 = 0$ and $\hat{u}_2 = 1$. Increasing the number of iterations improves the soft outputs associated with the information bits.

Second iteration

Row decoder:

$$\begin{aligned}\Lambda_{e1}(u_1) &= \Lambda(r_{11}) + \Lambda_{e2}(u_1) = +2.5 - 0.42 = +1.08,\\ \Lambda_{e1}(u_2) &= \Lambda(r_{12}) + \Lambda_{e2}(u_2) = +1.0 + 0.38 = +1.38.\end{aligned}$$

+0.58	+2.5
-2.12	+1.0
+0.5	X

Column decoder:

$$\Lambda_{c2}(u_1) = \Lambda(u_2 + r_{21}) = \log\left[\frac{1+e^{-1.62}}{e^{-2.12}+e^{+0.5}}\right] = -0.17,$$

$$\Lambda_{e2}(u_2) = \Lambda(u_1 + r_{21}) = \log\left[\frac{1+e^{+1.08}}{e^{+0.58}+e^{+0.5}}\right] = +0.16.$$

A-posteriori LLR:

$$\begin{aligned}\Lambda(u_1) &= \Lambda_{c1} + \Lambda_{e1}(u_1) + \Lambda_{e2}(u_1) = +0.41,\\ \Lambda(u_2) &= \Lambda_{c2} + \Lambda_{e1}(u_2) + \Lambda_{e2}(u_2) = -1.96.\end{aligned}$$

After the second iteration, based on the signs of the LLRs, the decoder delivers the same (correct) hard decisions. Moreover, note that the soft outputs changed because of the influence of the extrinsic LLR computed in the first iteration. In particular, the reliability of the first bit (the amplitude of the LLR) decreases from $|\Lambda(u_1)| = +1.58$ to $|\Lambda(u_1)| = +0.41$.

Error performance

With reference to Figure 78, page 144, it is clear that, as the number of iterations grows, the error performance improves. Typical of turbo coding schemes is the fact that increasing the number of iterations results in a monotonically decreasing improvement in coding gain. From Figure 78, increasing the number of iterations from 2 to 6 gives an improvement in SNR of 1.7 dB, whereas going from 6 to 18 iterations yields only a 0.3 dB improvement in coding gain.

Since the appearance of turbo codes, advances have taken place in understanding the BER behavior of turbo codes. At the time of this writing, there appears to be a consensus among researchers on the reasons why turbo codes offer such an excellent error performance:

- Turbo codes have a weight distribution that approaches, for long interleavers, that of random codes.
- Recursive convolutional encoders and proper interleaving map most of the low-weight information sequences into higher weight coded sequences.
- Systematic encoders allow the effective use of iterative techniques utilizing MAP decoders. Information symbol estimates are available directly from the channel.

In addition, most recently (end of 2001), a number of interesting semi-analytical tools have appeared, based on density evolution [RU], Gaussian approximation [EH] and mutual information [tBr2] or SNR transfer [DDP] characteristics, to study the convergence properties of iterative decoding algorithms.

The art of interleaving

A critical component in achieving good performance with iterative decoding of a turbo code is the interleaver. In turbo codes, the interleaver serves three main purposes: (1) Build very long codes with weight distributions that approach those of random codes; and (2) help in the iterative decoding process by decorrelating the input LLRs to the SISO decoders as much as possible; and (3) proper termination of the trellis in a known state, after the transmission of short to medium length frames, to avoid *edge effects* that increase the multiplicity of low weight paths in the trellises of the component codes. To emphasize, the specific type of interleaver becomes an important factor to consider as the frame lengths (or interleaver lengths) become relatively small, say, up to one thousand symbols. There is already a wealth of publications devoted to interleaver design for turbo codes. In this section, a brief description of the basic interleaver types and pointers to the literature are given.

In 1970, several types of optimum interleavers were introduced [Ram]. In particular, an (n_1, n_2) interleaver was defined there as a device that *"reorders a sequence so that no contiguous sequence of n_2 symbols in the reordered sequence contains any symbols that were separated by fewer than n_1 symbols in the original ordering."*

Let $\cdots, a_{\ell_1}, a_{\ell_2}, a_{\ell_3}, \cdots$ denote the output sequence from an (n_1, n_2) interleaver, where $\ell_1, \ell_2, \cdots$ are the positions of these symbols in the input sequence. Then the definition in the previous paragraph translates into the following condition:

$$|\ell_i - \ell_j| \geq n_1,$$

whenever

$$|i - j| < n_2.$$

It can be shown that the deinterleaver of an (n_1, n_2) interleaver is itself an (n_1, n_2) interleaver. The delay of the interleaver and deinterleaver are both equal to the delay introduced by the overall interleaving-deinterleaving operation. Another important parameter of an interleaver is the amount of storage or memory required. Ramsey showed four types of (n_1, n_2) interleavers that are optimum in the sense of minimizing the delay and memory required to implement them. These interleavers are known as *Ramsey interleavers*. At about the same time, Forney [For5] proposed an interleaver with the same basic structure of an (n_1, n_2) Ramsey interleaver, known as a *convolutional interleaver*. Convolutional interleavers were discussed in Section 6.2.3 and have been applied to the design of good turbo coding schemes [HaW].

There are several novel approaches to the analysis and design of interleavers: One is based on a *random interleaver* with a *spreading property*. Such a structure was first proposed in [DP], together with a simple algorithm to construct *S-random interleavers*: Generate random integers in the range $[1, N]$, and use a constraint on the *interleaving distance*. This constraint is seen to be equivalent to the definition of a Ramsey $(S_2, S_1 + 1)$ interleaver (this is noted in [Vuc], pp. 211-213). Then additional constraints (e.g., based on the empirical correlation between successive extrinsic LLR values) are imposed to direct the selection of the positions of the permuted symbols at the output of the interleaver [HEM, SSSN]. A second approach to the design of interleavers is to consider the overall turbo encoder structure and to compute its minimum distance and error coefficient (the number of coded sequences at minimum distance) [GPB, BH]. This gives an accurate estimation of the error floor in the medium to high SNR region. Other important contributions to the design of short interleavers for turbo codes are [TC, BP].

8.2.2 *Serial concatenation*

Serial concatenation of codes was introduced in [BDMP1]. A block diagram of an encoder of a serial concatenation of two linear codes is shown in Figure 83. Based on the results from Section 6.2.3, and in particular comparing Figure 83 with Figure 55, the serial concatenation of two linear codes is easily recognized as a *product code*. Note that, as opposed to a turbo code, in a serially concatenated coding system there is no puncturing of redundant symbols.

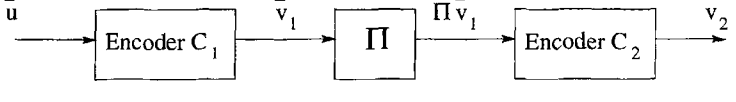


Figure 83 Block diagram of the encoder of a serially concatenated code.

The encoder of a serial concatenation of codes has the same structure as that of a product code. Following closely the notation in the original paper [BDMP1], the outer (p, k, d_1) code C_1 has a rate $R_1 = k/p$ and the inner (n, p, d_2) code C_2 has a rate $R_2 = p/n$. The codes are connected in the same manner as a block product code, with a block interleaver of length $L = mp$. This is achieved, as before, by writing m codewords of length p into the interleaver, and reading in a different order according to the permutation matrix Π. The sequence of L bits at the output of the interleaver is sent in blocks of p bits to the outer encoder. The rate of the overall $(N, K, d_1 d_2)$ code C_{SC} is $R_{SC} = k/n$, where $N = nm$ and $K = km$.

The generator matrix of C_{SC} can be expressed as the product of the generator matrix of C_1, the $k_2 \times n_1$ permutation matrix Π of the interleaver, and the generator matrix of C_2, as

follows

$$G_{\mathrm{SC}} = \begin{pmatrix} G_1 & & & \\ & G_1 & & \\ & & \ddots & \\ & & & G_1 \end{pmatrix} \Pi \begin{pmatrix} G_2 & & & \\ & G_2 & & \\ & & \ddots & \\ & & & G_2 \end{pmatrix} = (\, G_1' \mid \Pi \mid G_2' \,), \quad (8.12)$$

where G_i is the generator matrix of code C_i, $i = 1, 2$. The number of times that G_1 appears in the first factor G_1' of G_{SC} in (8.12) is k_2, and the number of times that G_2 appears in the third factor G_2' of G_{SC} is n_1. All other entries in G_1' and G_2' are zero.

Example 91 Let C_1 and C_2 be the same codes as in Example 89, that is, binary repetition $(2, 1, 2)$ and SPC $(3, 2, 2)$ codes, respectively. Then the serial concatenation, or product, of C_1 and C_2, C_{SC} is a binary linear block $(6, 2, 4)$ code. Note that the minimum distance of C_{SC} is larger than that of C_{PC} in Example 89. The generator matrices are $G_1 = (\,1 \quad 1\,)$ and $G_2 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$. Assuming that a conventional product code is employed, the permutation matrix, associated with a row-by-row and column-by-column interleaver, is

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Therefore, the generator matrix of C_{SC} is

$$\begin{aligned} G_{\mathrm{SC}} &= \begin{pmatrix} (1 \;\; 1) & \bar{0}_{12} \\ \bar{0}_{12} & (1 \;\; 1) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} & \bar{0}_{23} \\ \bar{0}_{23} & \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} & \bar{0}_{23} \\ \bar{0}_{23} & \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} (1 & 0 & 1) & (1 & 0 & 1) \\ (0 & 1 & 1) & (0 & 1 & 1) \end{pmatrix} \\ &= \left(\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \;\; \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \right), \end{aligned}$$

where $\bar{0}_{ij}$ denotes the $i \times j$ all-zero matrix.

The result can be verified by noticing that the last equality contains the generator matrix of the the SPC $(3, 2, 2)$ code *two times*, because of the repetition $(2, 1, 2)$ code. It is also interesting to note that this is the smallest member of the family of *repeat-and-accumulate codes* [DJM].

Note that the main difference between the serial concatenated coding scheme and product coding discussed in Section 6.2.3, is that there the interleaver was either a row-by-row column-by-column interleaver or a cyclic interleaver (if the component codes were cyclic). In contrast, as with turbo codes, the good performance of serial concatenation schemes generally depends on an interleaver that is chosen as "random" as possible.

Contrary to turbo codes, serially concatenated codes do not exhibit "interleaver gain saturation" (i.e., there is no error floor). Using a random argument for interleavers of length N, it can be shown that the error probability for a product code contains a factor $N^{-\lfloor (d_{Of}+1)/2 \rfloor}$, where d_{Of} denotes the minimum distance of the outer code, as opposed to a factor N^{-1} for parallel concatenated codes [BDMP1].

As a result, product codes outperform turbo codes in the SNR region where the error floor appears. At low SNR values, however, the better weight distribution properties of turbo codes [PSC] leads to better performance than product codes.

The following design rules were derived for the selection of component codes in a serially concatenated coding scheme[6] for component convolutional codes:

- The inner code must be an RSC code.
- The outer code should have large and, if possible, odd value of minimum distance.
- The outer code may be a nonrecursive (FIR) nonsystematic convolutional encoder.

The last design criterion is needed in order to minimize the number of codewords of minimum weight (also known as the *error exponent*) and the weight input sequences resulting in minimum weight codewords.

Iterative decoding of serially concatenated codes

With reference to Figure 84, note that if the outer code is a nonsystematic convolutional, then it is not possible to obtain the extrinsic information from the SISO decoder [BDMP1]. Therefore, different from the iterative decoding algorithm for turbo codes, in which only the LLR of the information symbols are updated, here the LLR of both information and code symbols are updated. The operation of the SISO decoder for the inner code remains unchanged. However, for the outer SISO decoder, the a-priori LLR is always set to zero, and the LLR of both information and parity symbols is computed and delivered, after interleaving, to the SISO decoder for the inner code as a-priori LLR for the next iteration. As with iterative decoding of turbo codes, there is a max-log-MAP based iterative decoding algorithm, as well as a version of SOVA that can be modified to become an approximated MAP decoding algorithm for iterative decoding of product codes [FV].

8.2.3 Block product codes

Although turbo codes and serial concatenations of RSC codes seem to have dominated the landscape of coding schemes where iterative decoding algorithms are applied, block product codes may also be used, as is evident from the foregoing discussion. In 1993, at the same conference where Berrou and colleagues introduced turbo codes, a paper was presented on iterative decoding of product and concatenated codes [LYHH]. In particular, a three-

[6] It should be noted that these criteria were obtained based on *union bounds* on the probability of a bit error.

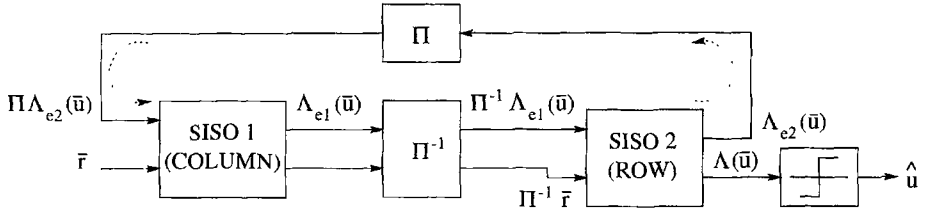


Figure 84 Block diagram of an iterative decoder for a serially concatenated code.

dimensional product $(4096, 1331, 64)$ code, based on the extended Hamming $(16, 11, 4)$ code, with iterative MAP decoding was considered and shown to achieve impressive performance.

One year later, near-optimum turbo-like decoding of product codes was introduced in [PGPJ] (see also [Pyn]). There the product of linear block codes of relatively high rate, single- and double-error correcting extended BCH codes, was considered. An iterative decoding scheme was proposed where the component decoders use the Chase type-II algorithm[7]. After a list of candidate codewords is found, LLR values are computed. This iterative decoding algorithm and its improvements are described in the next section.

Iterative decoding using Chase algorithm

In [PGPJ, Pyn], the Chase type-II decoding algorithm is employed to generate a list of candidate codewords which are close to the received word. Extrinsic LLR values are computed based on the best two candidate codewords. If only one codeword is found, then an approximated LLR value is output by the decoder. Let C be a binary linear (N, K, d) block code, capable of correcting any combination of $t = \lfloor (d-1)/2 \rfloor$ or less random bit errors. Let $\bar{r} = (r_1, r_2, \cdots, r_N)$ be the received word from the output of the channel, $r_i = (-1)^{v_i} + w_i$, $\bar{v} \in C$, where w_i is a zero-mean Gaussian random variable with variance $N_0/2$. Chase type-II algorithm is executed based on the received word $\bar{r}$, as described on page 130.

Three possible events can happen at the end of the Chase type-II algorithm:

1. Two or more codewords, $\{\hat{v}_1, \cdots, \hat{v}_\ell\}$, $\ell \geq 2$, are found;
2. One codeword $\hat{v}_1$ is found; or
3. No codeword is found.

In the last event, the decoder may raise an uncorrectable error flag and output the received sequence as is. Alternatively, the number of error patterns to be tested can be increased until a codeword is found, as suggested in [Pyn].

Let $\mathcal{X}_j(\ell)$ denote the set of modulated codewords of C, found by Chase algorithm, for which the j-th component $x_j = \ell$, $\ell \in \{-1, +1\}$, for $1 \leq j \leq N$. Denote by $\bar{x}_j(\ell), \bar{y}_j(\ell) \in \mathcal{X}_j(\ell)$ the closest and next closest modulated codewords to the received word $\bar{r}$, in the Euclidean distance sense, respectively.

By using the log-max approximation $\log(e^a + e^b) \approx \max(a, b)$, the symbol LLR value (8.2) can be expressed as [Pyn, FL5]

$$\Lambda(u_j) \sim \Lambda'(u_j) = \frac{4E}{N_0} \left(|\bar{r} - \bar{y}_j(-1)|^2 - |\bar{r} - \bar{x}_j(+1)|^2 \right), \tag{8.13}$$

[7] Chase algorithms are discussed in Section 7.4.

from which, after normalization and redefining $x_m = x_m(+1)$ and $y_m = y_m(-1)$, the *soft output* is

$$\Lambda'(u_j) = r_j + \sum_{\substack{x_m \neq y_m \\ m=1,\, m\neq j}}^{N} r_m x_m = x_j \sum_{\substack{x_m \neq y_m \\ m=1}}^{N} r_m x_m. \tag{8.14}$$

The term

$$w_j = x_j \sum_{\substack{x_m \neq y_m \\ m=1, m\neq j}}^{N} r_m x_m \tag{8.15}$$

is interpreted as a *correction term* to the soft-input r_j, which depends on the two modulated codewords closest to $\bar{r}$, and plays the same role as the *extrinsic LLR*. For $1 \leq j \leq N$, and each position j, the value w_j is sent to the next decoder as extrinsic LLR, with a *scaling factor* α_c, so that

$$r'_j = r_j + \alpha_c w_j, \tag{8.16}$$

is computed as the soft input at the next decoder. The factor α_c is used to compensate for the difference in the variances of the Gaussian random variables r_i and r'_j. A block diagram of the procedure for the generation of soft input values and extrinsic LLR values is shown in Figure 85.

If for a given j-th position, $1 \leq j \leq N$, no pair of sequences $\bar{x}_j(+1)$ and $\bar{y}_j(-1)$ can be found by Chase algorithm, in [Pyn] it has been suggested the use of the following symbol LLR,

$$\Lambda'(u_j) = \beta_c x_j, \tag{8.17}$$

where β_c is a *correction factor* to compensate for the approximation in the extrinsic information, and was estimated by simulations as

$$\beta_c = \left| \log \left(\frac{\Pr\{v_j \text{ correct}\}}{\Pr\{v_j \text{ incorrect}\}} \right) \right|, \tag{8.18}$$

that is, the magnitude of the LLR of the simulated symbol error rate. In [PP, MT], it is shown how the correction factors α and β can be computed *adaptively* based on the statistics of the processed codewords. It should also be noted that the soft-output algorithm proposed in [FL5], and described in Section 7.5, can also be applied. Adaptive weights are also needed in this case, to scale down the extrinsic LLR values.

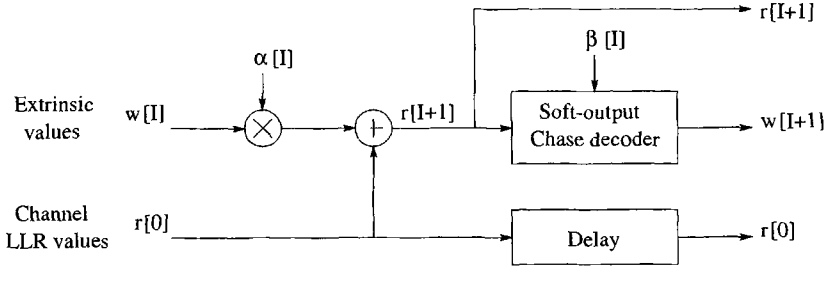


Figure 85 Block diagram of a soft-output Chase decoder.

Summarizing, the iterative decoding method with soft-outputs based on a set of codewords produced by a Chase type-II algorithm is as follows:

Step 0: Initialization

Set iteration counter $I = 0$. Let $\bar{r}[0] = \bar{r}$ (the received channel values).

Step 1: Soft inputs

For $j = 1, 2, \cdots, N$,

$$r_j[I+1] = r_j[0] + \alpha_c[I]\, w_j[I].$$

Step 2: Chase algorithm

Execute Chase type-II algorithm, using $\bar{r}[I+1]$. Let n_c denote the number of codewords found. If possible, save the closest two modulated codewords, $\bar{x}$ and $\bar{y}$ to the received sequence.

Step 3: Extrinsic information

For $j = 1, 2, \cdots, N$,

- If $n_c \geq 2$

$$w_j[I+1] = x_j \sum_{\substack{x_m \neq y_m \\ m=1, m \neq j}}^{N} r[I+1]_m x_m,$$

- Else

$$w_j[I+1] = \beta_c[I] x_j.$$

Step 4: Soft output

Let $I = I + 1$. If $I < I_{\max}$ (the maximum number of iterations) or a stopping criterion is not satisfied then go to Step 1.

Else compute the soft output,

For $j = 1, 2, \cdots, N$,

$$\Lambda(u_i) = \alpha_c[I]\, w_j[I] + r_j[0], \tag{8.19}$$

and stop.

For BPSK modulation, the values of α_c and β_c were computed for up to four iterations (eight values in total, two values for the first and second decoders), as [Pyn]

$$\alpha_c = (0.0 \quad 0.2 \quad 0.3 \quad 0.5 \quad 0.7 \quad 0.9 \quad 1.0 \quad 1.0),$$

$$\beta_c = (0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0).$$

8.3 Low-density parity-check codes

In 1962, Gallager in [Gal] introduced a class of linear codes, known as low-density parity-check (LDPC) codes, and presented two iterative probabilistic decoding algorithms. Later, Tanner [Tan] extended Gallager's probabilistic decoding algorithm to the more general case where the parity-checks are defined by subcodes, instead of simple single parity-check equations. Earlier, it was shown that LDPC codes have a minimum distance that grows linearly with the code length and that errors up to the minimum distance could be corrected with a decoding algorithm with almost linear complexity [ZP].

In [MN, Mac] it is shown that LDPC codes can get as close to the Shannon limit as turbo codes. Later in [RSU], irregular LDPC codes were shown to outperform turbo codes of approximately the same length and rate, when the block length is large. At the time of writing, the best rate-1/2 binary code, with a block length of 10,000,000, is an LDPC code that achieved a record 0.0045 dB away from the Shannon limit for binary transmission over an AWGN channel [CFRU].

A *regular* LDPC code is a linear (N, K) code with parity-check matrix H having the Hamming weight of the columns and rows of H is equal to J and K, respectively, with both J and K much smaller than the code length N. As a result, an LDPC code has a very sparse parity-check matrix. If the Hamming weights of the columns and rows of H are chosen in accordance to some nonuniform distribution, then *irregular* LDPC codes are obtained [RSU]. MacKay has proposed several methods to construct LDPC matrices by computer search [Mac].

8.3.1 Tanner graphs

For every linear (N, K) code C, there exists a bipartite graph with incidence matrix H. This graph is known as a *Tanner graph* [Tan, SS], named after its inventor. By introducing state nodes, Tanner graphs have been generalized to factor graphs [For7, KFL]. The nodes of the Tanner graph of a code are associated with two kinds of variables, and their LLR values.

The Tanner graph of a linear (N, K) code C has N *variable nodes* or *code nodes*, x_ℓ, associated with code symbols, and at least $N - K$ *parity nodes*, z_m, associated with the parity-check equations. For a regular LDPC code, the degrees of the code nodes are all equal to J and the degrees of the parity nodes equal to K.

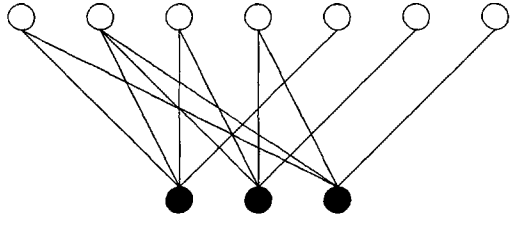


Figure 86 Tanner graph of a Hamming $(7, 4, 3)$ code.

Example 92 To illustrate the Tanner graph of a code, consider the Hamming $(7, 4, 3)$ code.

Its parity-check matrix is[8]

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

The corresponding Tanner graph is shown in Figure 86. The way code nodes connect to check nodes is dictated by the rows of the parity-check matrix.

The first row gives the parity-check equation, $v_1 + v_2 + v_3 + v_5 = 0$. As indicated before, variables x_ℓ and z_m are assigned to each code symbol and each parity-check equation, respectively. Therefore, the following parity-check equations are obtained,

$$\begin{aligned} z_1 &= x_1 + x_2 + x_3 + x_5, \\ z_2 &= x_2 + x_3 + x_4 + x_6, \\ z_3 &= x_1 + x_2 + x_4 + x_7. \end{aligned}$$

From the topmost equation, code nodes x_1, x_2, x_3 and x_5 are connected to check node z_1. Similarly, the columns of H, when interpreted as incidence vectors, indicate in which parity-check equations code symbols appear, or *participate* in. The leftmost column of H above, $(1 \quad 0 \quad 1)^\intercal$, indicates that x_1 is connected to check nodes z_1 and z_3.

Example 93 The parity-check matrix in Gallager's paper [Gal],

$$H = \begin{pmatrix} 1111\,0000\,0000\,0000\,0000 \\ 0000\,1111\,0000\,0000\,0000 \\ 0000\,0000\,1111\,0000\,0000 \\ 0000\,0000\,0000\,1111\,0000 \\ 0000\,0000\,0000\,0000\,1111 \\ \\ 1000\,1000\,1000\,1000\,0000 \\ 0100\,0100\,0100\,0000\,1000 \\ 0010\,0010\,0000\,0100\,0100 \\ 0001\,0000\,0010\,0010\,0010 \\ 0000\,0001\,0001\,0001\,0001 \\ \\ 1000\,0100\,0001\,0000\,0100 \\ 0100\,0010\,0010\,0001\,0000 \\ 0010\,0001\,0000\,1000\,0010 \\ 0001\,0000\,1000\,0100\,1000 \\ 0000\,1000\,0100\,0010\,0001 \end{pmatrix},$$

is that of an LDPC $(20, 5)$ code with $J = 3$ and $K = 4$. Its Tanner graph is shown in Figure 87. Note that every code node is connected to exactly three parity-check nodes. In other words, the *degree* of the code nodes is equal to $J = 3$. Similarly, the degree of the parity nodes is equal to $K = 4$.

Tanner graphs can be used to estimate codewords of an LDPC code C with iterative probabilistic decoding algorithms, based on either hard or soft decisions. In the following, the two basic iterative decoding algorithms introduced by Gallager are presented.

---

[8] See Example 13 on page 23.

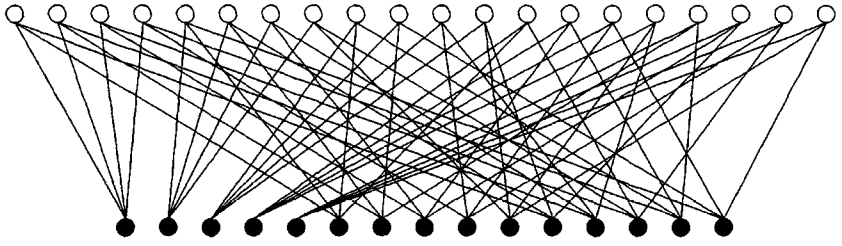


Figure 87 Tanner graph of Gallager's $(20, 5)$ code.

8.3.2 Iterative hard-decision decoding: The bit-flip algorithm

In his 1962 paper, Gallager gave the following algorithm [Gal][9].

> *The decoder computes all the parity checks and then changes any digit that is contained in more than some fixed number of unsatisfied parity-check equations. Using these new values, the parity-checks are recomputed, and the process is repeated until the parity checks are all satisfied.*

The input of the algorithm is the vector $\bar{r}_h = (\text{sgn}(r_1), \text{sgn}(r_2), \cdots, \text{sgn}(r_N))$, where $r_i = (-1)^{v_i} + w_i$, $\bar{v} \in C$, w_i denotes a zero-mean Gaussian random variable with variance $N_0/2$, $i = 1, 2, \cdots, N$, and $\text{sgn}(x) = 1$, if $x < 0$, and $\text{sgn}(x) = 0$ otherwise.

Let T denote a threshold such that, if the number of unsatisfied parity-check equations where a code symbol v_i participates exceeds T, then symbol is "flipped", $v_i = v_i \oplus 1$.

Figures 88 and 89 show simulation results for Gallager's $(20, 5)$ code C_G and the Hamming $(7, 4, 3)$ code C_H, respectively, with binary transmission over an AWGN channel. The maximum number of iterations was set to 4 for C_G and 2 for C_H. In both figures, the threshold was set to $T = 1, 2, 3$. For the Hamming code, in Figure 89, also shown are single-error hard-decision decoding (HD), soft-decision MLD decoding (SD) and the union bound (1.35).

In Figures 90 to 92, the error performance of the Berlekamp-Massey (BM) algorithm and Gallager bit-flip (BF) algorithm is compared for the BCH $(31, 26, 3)$, $(31, 21, 5)$ and $(31, 16, 7)$ codes, respectively. It is evident that, as the error correcting capability of the code increases, the performance of the BF algorithm is inferior to that of the BM algorithm. On the other hand, in terms of computational complexity, the Gallager BF algorithm requires simple exclusive-or gates and comparisons, as opposed to $GF(2^m)$ arithmetic processors in the case of the BM algorithm. This suggests that, for some high-rate codes, such as single- and double-error correcting BCH codes, the bit-flip algorithm might be considered an alternative to the BM algorithm.

An additional feature of the BF algorithm, as well as the iterative probabilistic decoding algorithm presented below, is that its complexity depends only on the degrees of the nodes in the Tanner graph. In other words, for fixed values of J and K, the decoding complexity grows linearly with the code length.

---

[9] This algorithm is also known as Gallager's algorithm A.

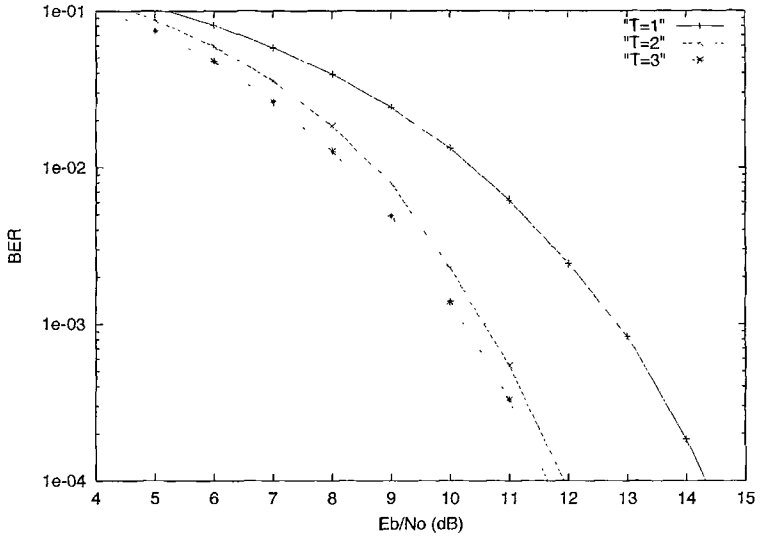


Figure 88 Performance of Gallager's $(20, 5)$ code with bit-flip decoding. Four iterations with different threshold values

8.3.3 Iterative probabilistic decoding: belief propagation

In this section, an iterative *belief-propagation* (IBP) decoding algorithm is presented. This algorithm is also known as *Pearl's algorithm* [Prl], and *sum-product algorithm* [Wib, KFL, For7]. Wiberg in his Ph.D. thesis [Wib] has shown that the forward-backward, turbo and Gallager algorithm (described below) are all special cases of the sum-product algorithm. Furthermore, it was demonstrated [MMC] that iterative decoding algorithms for turbo codes and product codes are particular cases of IBP decoding. The description below follows closely [Prl, Mac] for binary LDPC codes. The following notation is convenient in describing the algorithm. Let $h_{i,j}$ denote the entry of H in the i-th row and j-th column. Let

$$\mathcal{L}(m) = \{\ell : h_{m,\ell} = 1\}, \tag{8.20}$$

denote the set of *code positions* that participate in the m-th parity-check equation, and let

$$\mathcal{M}(\ell) = \{m : h_{m,\ell} = 1\}, \tag{8.21}$$

denote the set of *check positions* in which code position ℓ participates.

The algorithm iteratively computes two types of conditional probabilities:

$q^x_{m\ell}$ The probability that the ℓ-th bit of $\bar{v}$ has the value x, given the information obtained via the check nodes other than check node m.

$r^x_{m\ell}$ The probability [10] that a check node m is satisfied when bit ℓ is fixed to a value x and the other bits are independent with probabilities $q_{m\ell'}$, $\ell' \in \mathcal{L}(m) \setminus \ell$

[10] Note: This is an abuse of notation, because r_i denotes the i-th component of the received vector. However, it helps to keep the same notation as most of the literature in the topic.

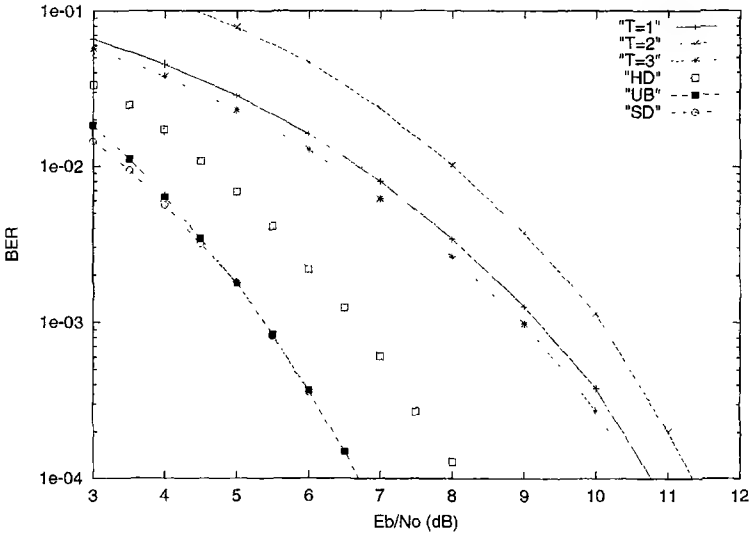


Figure 89 Performance of a Hamming $(7, 4, 3)$ code with bit-flip decoding. Two iterations with different threshold values.

As noted in [Prl, Mac], the following IBP decoding algorithm would produce the exact a-posteriori probabilities after some number of iterations, *if the Tanner graph of the code contained no cycles.* In the following, binary transmission over an AWGN channel is assumed. As before, the modulated symbols $m(v_i)$ are transmitted over an AWGN channel and received as $r_i = m(v_i) + w_i$, where w_i is a Gaussian distributed random variable with zero mean and variance $N_0/2, 1 \leq i \leq N$.

Initialization

For $\ell \in \{1, 2, \cdots, N\}$, initialize the a-priori probabilities of the code nodes,

$$p_\ell^1 = \frac{1}{1 + \exp(r_\ell \frac{4}{N_0})}, \tag{8.22}$$

and $p_\ell^0 = 1 - p_\ell^1$. For every (ℓ, m) such that $h_{m,\ell} = 1$,

$$q_{m,\ell}^0 = p_\ell^0, \qquad q_{m,\ell}^1 = p_\ell^1. \tag{8.23}$$

Message passing

Step 1: Bottom-up (horizontal):

For each ℓ, m, compute

$$\delta r_{m,\ell} = \prod_{\ell' \in \mathcal{L}(m) \setminus \ell} (q_{m,\ell'}^0 - q_{m,\ell'}^1), \tag{8.24}$$

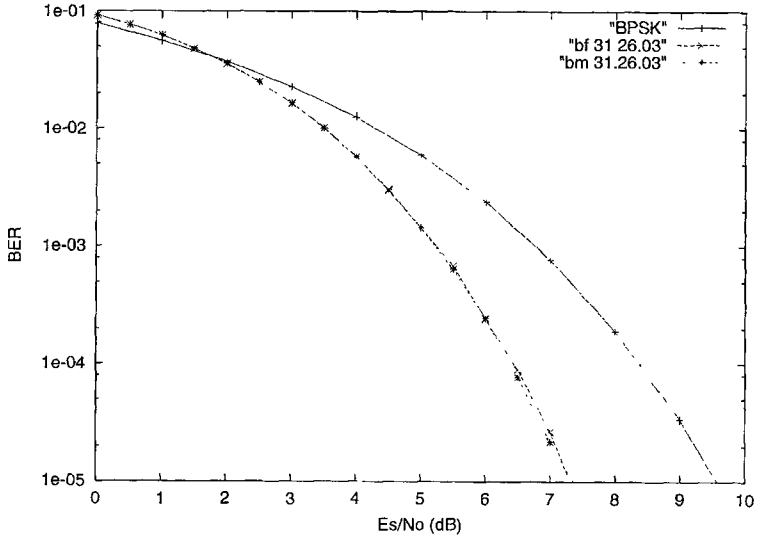


Figure 90 Performance of Berlekamp-Massey and Gallager bit-flip algorithms for the binary BCH $(31, 26, 3)$ code.

and

$$r^0_{m,\ell} = (1 + \delta r_{m,\ell})/2, \qquad r^1_{m,\ell} = (1 - \delta r_{m,\ell})/2. \tag{8.25}$$

Step 2: Top-down (vertical):

For each ℓ, m, compute

$$q^0_{m,\ell} = p^0_\ell \prod_{m' \in \mathcal{M}(\ell)\setminus m} r^0_{m',\ell}, \qquad q^1_{m\ell} = p^1_\ell \prod_{m' \in \mathcal{M}(\ell)\setminus m} r^1_{m',\ell}. \tag{8.26}$$

and normalize, with $\alpha = 1/(q^0_{m\ell} + q^1_{m\ell})$,

$$q^0_{m,\ell} = \alpha\, q^0_{m,\ell}, \qquad q^1_{m,\ell} = \alpha\, q^1_{m,\ell}. \tag{8.27}$$

For each ℓ, compute the a-posteriori probabilities

$$q^0_\ell = p^0_\ell \prod_{m \in \mathcal{M}(\ell)} r^0_{m,\ell}, \qquad q^1_\ell = p^1_\ell \prod_{m \in \mathcal{M}(\ell)} r^1_{m,\ell}, \tag{8.28}$$

and normalize, with $\alpha = 1/(q^0_\ell + q^1_\ell)$,

$$q^0_\ell = \alpha\, q^0_\ell, \qquad q^1_\ell = \alpha\, q^1_\ell. \tag{8.29}$$

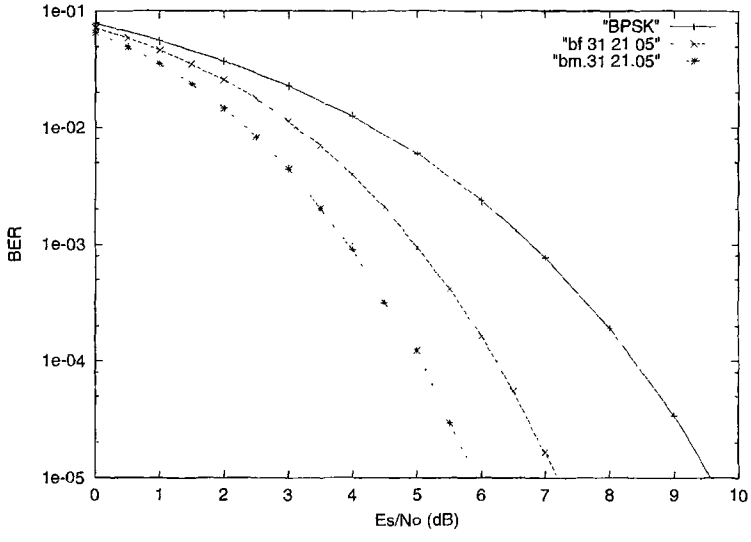


Figure 91 Performance of Berlekamp-Massey and Gallager bit-flip algorithms for the binary BCH $(31, 21, 5)$ code.

Decoding and soft-outputs

For $i = 1, 2, \cdots N$, compute

$$\hat{v}_i = \text{sgn}(q_i^0) \tag{8.30}$$

If $\hat{v}H^\top = \bar{0}$, then $\hat{v}$ is the estimated codeword and the soft outputs are

$$\Lambda(v_i) = \log(q_i^1) - \log(q_i^0), \quad 1 \leq i \leq N. \tag{8.31}$$

The algorithm stops.

Otherwise, return to Step 2. If the number of iterations exceeds a predetermined threshold, a decoding failure is declared. Output received values as they are. The algorithm stops.

Figure 93 shows the performance of IBP decoding for the binary cyclic PG (273,191,17) code. This code is a small member of the family of binary finite geometry LDPC codes recently introduced in [KLF]. Also shown in the plot is the performance of the BF algorithm with a threshold equal to 8.

Notes

As proposed in [Gal], the IBP decoding algorithm can be modified in a straightforward way to use log-likelihood ratios instead of probabilities. This has the advantage that no normalization is required in the second step of message passing. In this case, the function

$$F(x) \triangleq \frac{e^x + 1}{e^x - 1} = \frac{1}{\tanh(x/2)}, \tag{8.32}$$

or its inverse, is needed in Step 1 of message passing. A program implementing this log-IBP decoder is available on the ECC web site.

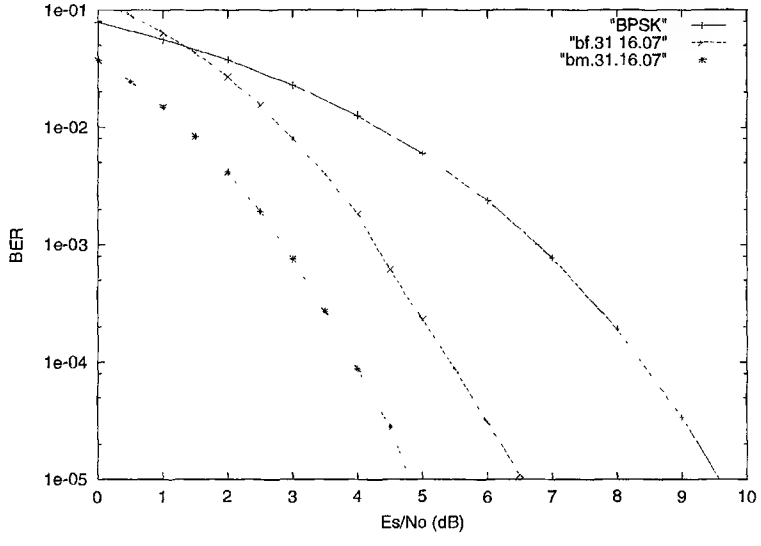


Figure 92 Performance of Berlekamp-Massey and Gallager bit-flip algorithms for the binary BCH $(31, 16, 7)$ code.

Note that $F(x)$ can be implemented as a look-up table. Numerical results suggest that quantization of $F(x)$ to eight levels results in practically no performance loss compared with a floating point version. In [FMH], several reduced-complexity versions of the IBP algorithm are presented that favorably trade off decoding complexity and error performance. Other interesting applications of iterative decoding algorithms include fast-correlation attacks in cryptanalysis [MG, Glc].

It is notable that LDPC codes can always detect whenever decoding errors occur, whereas turbo codes and product codes based on convolutional codes cannot detect many errors [Mac]. As it is evident from the foregoing presentation, LDPC codes have very low complexity compared with iterative decoding using component MAP decoders. This is true when complexity is measured in terms of the number of real additions and multiplications per block per iteration.

However, it should be pointed out that IBP decoding gives the MLD codeword *only if the Tanner graph contains no cycles*. Because for most practical codes the Tanner graph has relative short cycles (lengths 4 and 6), convergence of the IBP decoding algorithm is either not guaranteed or slow [RSU, KFL, For7]. As a result, in general, IBP decoding algorithms for LDPC codes may require much more iterations than iterative decoding algorithms for product codes with MAP component decoders.

In terms of implementation of IBP decoding, two architectures are the following. A fast parallel architecture can be realized with N X-processors for code nodes and M Z-processors for check nodes, with connections between them specified by multiple address computation units (ACU). This architecture is shown in Figure 94. In the figure, λ and π are LLR values associated with the conditional probabilities used in the IBP algorithm. Alternatively, only one X-processor and one Z-processor can be used and shared between nodes, whose metrics are stored in two memories, a π-memory and a λ-memory, as shown in Figure 95. These architectures represent the extremes in the space/time tradeoff to find the best architecture.

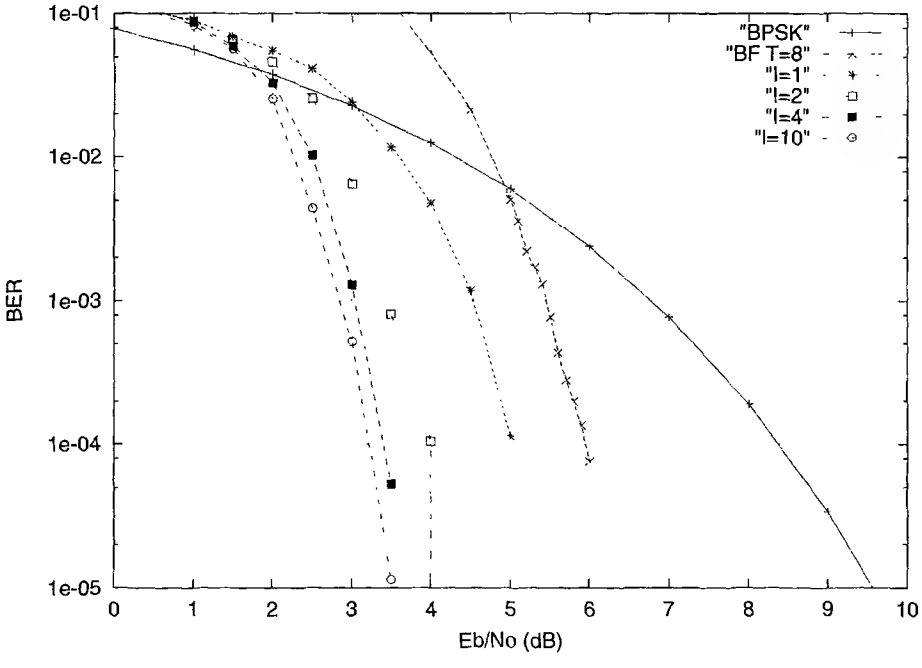


Figure 93 Performance of iterative decoding of a binary (273, 191) cyclic code.

The number of computations in the IBP decoding algorithm can be reduced by making preliminary *hard decisions* based upon the amplitudes of the log-likelihood ratio values (reliabilities) of the channel symbols. This idea appears in [FK] and in [Prl] as a method of dealing with short cycles in Bayesian networks (Tanner graphs). A set of highly reliable positions (HRP) can be selected by comparing the channel LLR values to a threshold T. If the reliability of a channel symbol exceeds T, then that code node is fixed to a hard decision (HD) value. As a result, the X-processor is not utilized in the decoding process. Instead, this processor always gives as an output either the maximum probability (LLR) or a flag indicating that this position is highly reliable. Consequently, the Z-processor to where the HD value is sent performs a smaller number of computations, because this input is highly reliable and not taken into account in the probability or LLR computations.

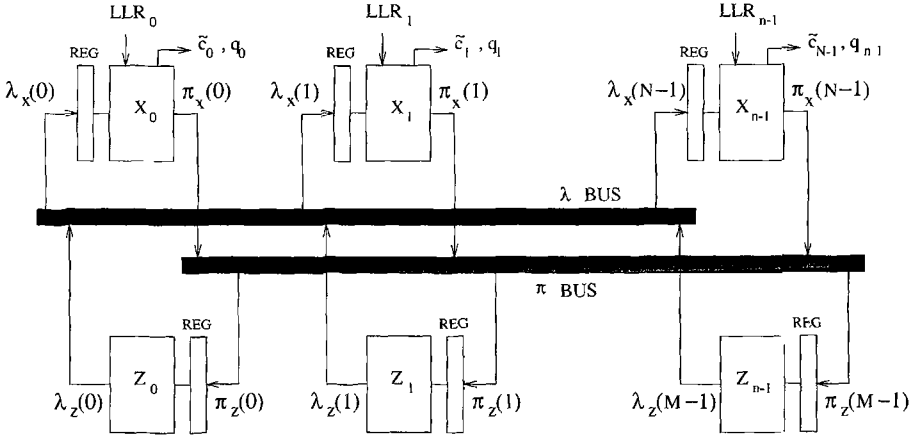


Figure 94 Parallel architecture of an IBP decoder.

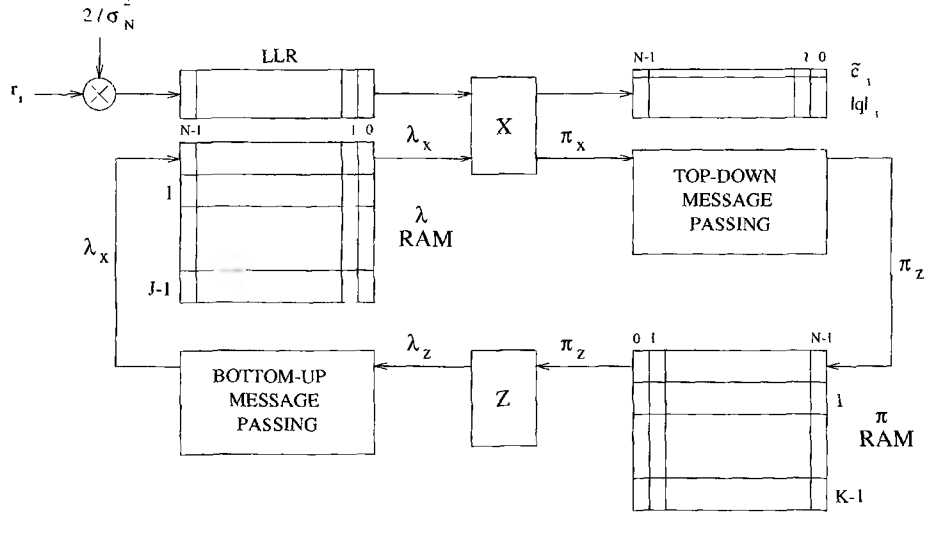


Figure 95 Serial architecture of an IBP decoder.

9

Combining codes and digital modulation

In discussing soft-decision decoding, attention was focused on binary transmission, i.e., using two possible transmitted symbols $\{-1, +1\}$. Let the *Nyquist bandwidth* of a transmission (or storage) signal be the rate R_s at which symbols are transmitted (or stored). For binary transmission, the two values of a bit, 0 and 1, are assigned to two values of the transmitted symbol, +1 and -1, respectively. Therefore, R_s bits/second require a Nyquist bandwidth of R_s Hz. The *spectral efficiency* μ of a binary transmission system equals 1 bit/sec/Hz (or 1 bit/symbol). Coded modulation is the joint design of error correcting codes and digital modulation formats to increase the bandwidth efficiency of a digital communication system.

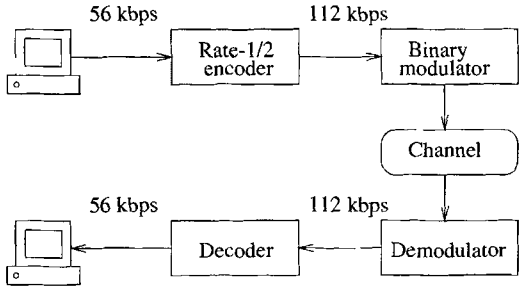
9.1 Motivation

Suppose that an error correcting coding scheme is required to increase the reliability of the binary transmission (or storage) system. Let $R_c = k/n$ denote the rate of the code. Then the spectral efficiency is $\mu = R_c$ bps/Hz. Thus for a coded binary transmission system a spectral efficiency $\mu \leq 1$ bps/Hz is achieved. This translates into a faster signalling rate or *larger bandwidth* for the same bit rate. Equivalently, the *bit rate has to be reduced*, by a factor of $1/R_s$, so as to keep the transmitted symbol rate or bandwidth constant.

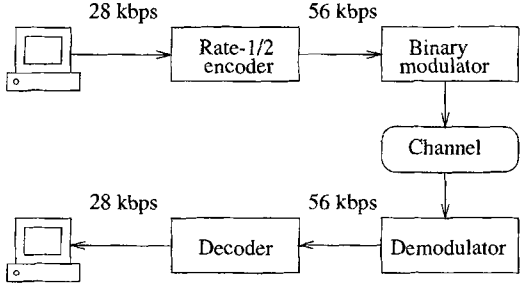
Example 94 Assume that 56 kbps are desired to be transmitted over a channel. The required Nyquist bandwidth of an uncoded binary transmission system is 56 kHz. Suppose a rate-1/2 convolutional code is used. Then, with binary transmission, the spectral efficiency of the coded system is $\mu = 0.5$ bps/Hz. The effect of μ in data rate and signalling rate is illustrated in Figure 96.

Increasing the bandwidth of the signal, as in Figure 96 (a) is not a practical solution, since typically the channel *bandwidth is expensive* (or limited, as in a telephone line). Decreasing data rate (Figure 96 (b)) is a possible solution, but places a limit on the number of services or applications offered. In addition, the *increased transmission delay* may not be acceptable (e.g., voice or video).

The fundamental question then is the following: How to increase the data rate without increasing the bandwidth (or symbol rate)? The answer, as given by Ungerboeck [Ung1] and Imai and Hirakawa [IH], is to use an expanded signal set (e.g., 2^ν-ary PSK or QAM digital modulation) and then to apply error correcting coding to increase the Euclidean distance between coded sequences.



(a) Keeping bit rate constant



(b) Keeping symbol rate constant

Figure 96 Effect of coding in *binary transmission* of information.

9.1.1 Examples of signal sets

Several signal sets used in digital communication systems are shown in Figure 97. The I and Q axis represent *orthogonal signals* that are used in transmission.[1] In digital communication systems, generally

$$I = \cos(2\pi f_c t), \qquad Q = \sin(2\pi f_c t), \tag{9.1}$$

where I stands for *in-phase* and Q for *quadrature*. If a point in the IQ-plane has coordinates (x, y), then the transmitted signal is

$$s(t) = R\cos(2\pi f_c t + \phi), \qquad (k-1)T \leq t < kT, \tag{9.2}$$

where $R = \sqrt{x^2 + y^2}$, $\phi = \tan^{-1}(y/x)$, and T is the symbol duration. In other systems (e.g., storage), orthogonal pulses may be used, such as those illustrated in Figure 98. The set of signal symbols in the two-dimensional IQ-plane is called a *constellation*, and symbols are called *signal points*.

From the viewpoint of digital signal processing, *modulation* is a *mapping*. That is, the process of assigning an ν-dimensional binary vector $\bar{b}$ to a signal point $(x(\bar{b}), y(\bar{b}))$ in the

[1] For simplicity of exposition, let $I \triangleq I(t)$ and $Q \triangleq Q(t)$.

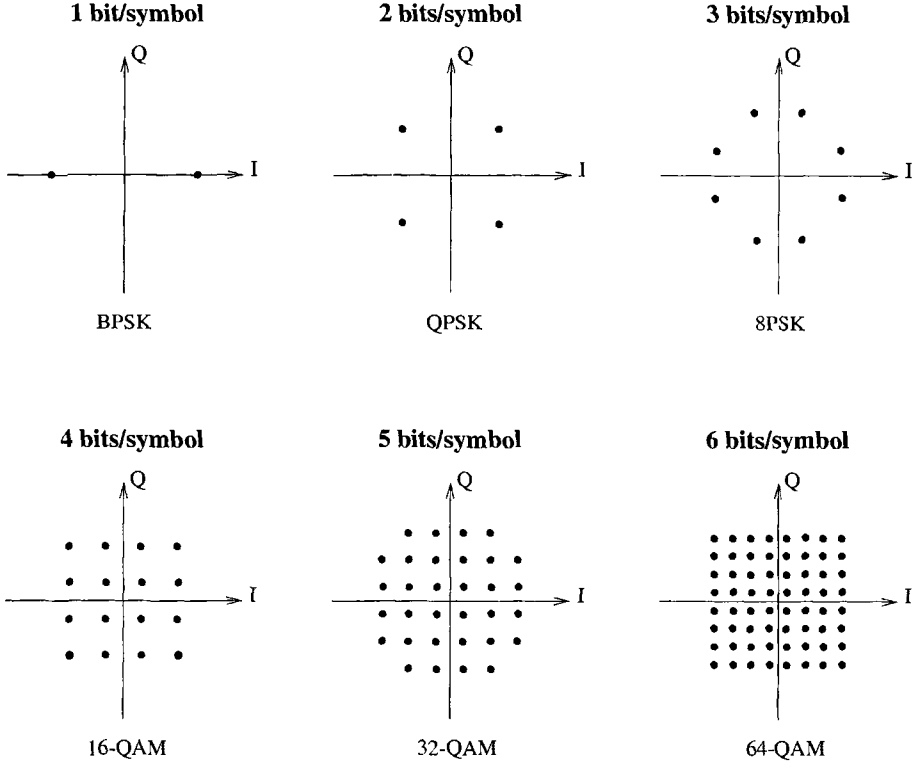


Figure 97 Examples of M-PSK and M-QAM signal constellations.

constellation. In previous chapters, only BPSK was considered, in which case $\nu = 1$. For $\nu > 1$, there are many possible assignments of bits to signal points. That is, many ways to *label the signal points*. Figure 99 shows an example of a QPSK constellation ($\nu = 2$) with two different (non-equivalent) mappings.

Moving from binary modulation to 2^ν-ary modulation has the advantage that the number of bits per symbol is increased by a factor of ν, thus *increasing the spectral efficiency* of the system. On the other hand, the required average energy of the signal increases (QAM), or the distance between modulation symbols decreases (PSK). In practice, transmitted power is limited to a maximum value. This implies that the *signal points become closer* to each other. Recall that the probability of error in an AWGN channel between two signal points separated by an Euclidean distance equal to D_e is [Hay. Pro]

$$\Pr(\epsilon) = Q\left(\sqrt{\frac{D_e^2}{2N_0}}\right), \tag{9.3}$$

where $Q(x)$ is given by (1.2). As a result, a *higher probability of error* is experienced at the receiver end. In this sense, the function of error correcting coding is to reduce the probability of error $\Pr(\epsilon)$ and to improve the quality of the system.

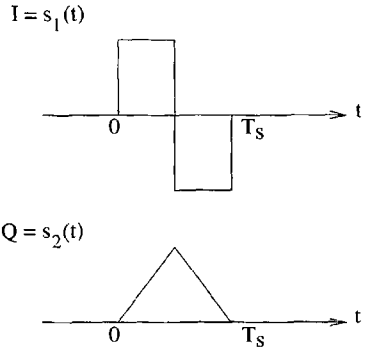
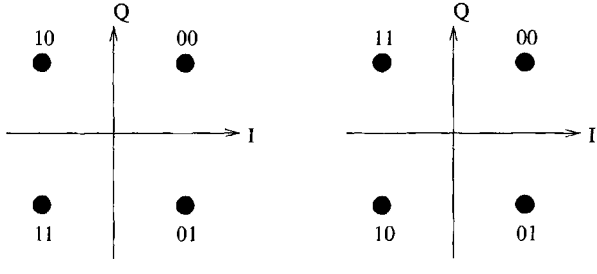


Figure 98 An example of two orthogonal pulses.



(a) GRAY MAP (b) NATURAL MAP

Figure 99 Two different mappings of a QPSK constellation.

9.1.2 Coded modulation

In 1974, Massey introduced the key concept of treating coding and modulation as a joint signal processing entity [Mas3], see Figure 100. That is, the *coordinated design* of error correcting coding and modulation schemes.

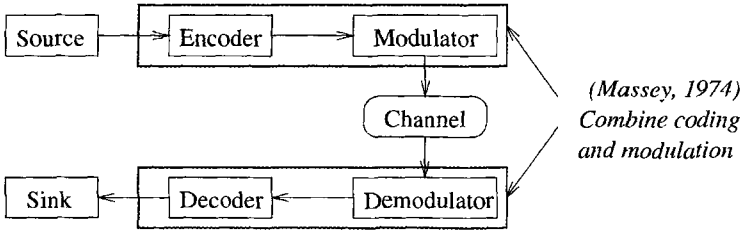


Figure 100 The idea of joint coding and modulation.

Two fundamental questions on combining coding and modulation arise:

1. How to construct the bits-to-symbols mapping?
2. How to assign coded *bit sequences* to coded *symbol sequences*?

Two basic approaches were proposed in the 1970s to design coded modulation systems:

1. *Trellis Coded Modulation (TCM) [Ung1]*
   Apply a natural mapping of bits to signals, through set partitioning. Given an underlying finite-state machine, assign symbol sequences to trellis paths. Perform *Viterbi decoding* at the receiver.

2. *Multilevel Coded Modulation (MCM) [IH]*
   Apply a mapping of codewords to *bit positions*, through a binary partition. For 2^ν-ary modulation, use ν error correcting codes, one per label bit. Perform *multistage decoding* at the receiver.

In both TCM and MCM, the basic idea is to *expand the constellation* in order to obtain the *redundancy* needed for error correcting coding, and then to use coding to increase the *minimum Euclidean distance* between sequences of modulated signals.

9.1.3 Distance considerations

To illustrate how error correcting codes and digital modulation can be combined, and the consequent increase in the minimum distance between signal sequences from an expanded signal constellation, consider the following.

Example 95 A block coded QPSK modulation scheme is shown in Figure 101. Codewords of the extended Hamming $(8, 4, 4)$ code are divided in four pairs of symbols and mapped to QPSK signal points with Gray mapping. A nice feature of Gray labeling of QPSK points is that the squared Euclidean distance between points is equal to twice the Hamming distance between their labels. As a result, a block coded modulation scheme with $\mu = 1$ bits/symbol and a *minimum squared Euclidean distance*, or MSED, $D^2_{\min} = 8$ is obtained. An uncoded system with the same spectral efficiency is BPSK, which has an MSED $D^2_{unc} = 4$. Consequently, the *asymptotic coding gain* of this scheme is

$$G = 10 \log_{10} \left(\frac{D^2_{\min}}{D^2_{unc}} \right) = 3 \text{ dB}. \tag{9.4}$$

Over an AWGN channel, this coded QPSK modulation requires half the power of uncoded BPSK modulation, to achieve the same probability of error $P(\epsilon)$. Figure 102 shows simulation results of this scheme.

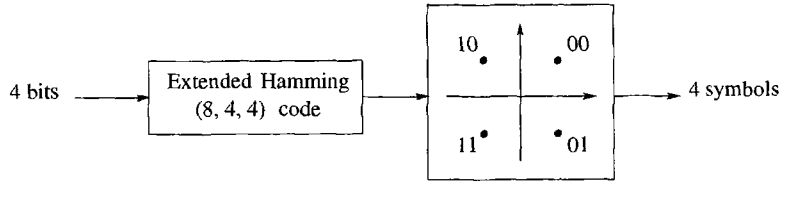


Figure 101 Block coded QPSK modulation using an extended Hamming $(8, 4, 4)$ code.

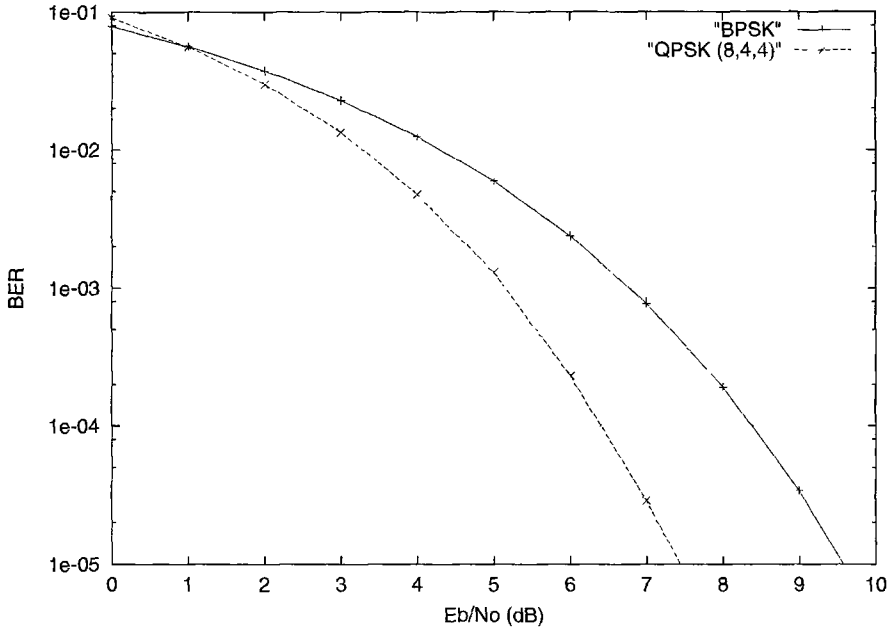


Figure 102 Simulation results of a block coded QPSK modulation using an extended Hamming $(8, 4, 4)$ code. AWGN channel.

Therefore, while the use of an expanded 2^ν-ary modulation causes the distance between signal point to decrease, a properly selected error correcting code can make sequences of signal points at a minimum distance larger than that of an uncoded system, with the same spectral efficiency.

9.2 Trellis-coded modulation (TCM)

Proposed by Ungerboeck in 1976, the main idea in TCM is to perform *mapping by set partitioning*. A basic trellis structure, associated with the state transitions of a *finite-state machine*, is selected and signal subsets mapped to trellis branches. For systems that require high spectral efficiency, uncoded bits may be assigned to parallel branches in the trellis.

9.2.1 Set partitioning and trellis mapping

Bit labels assigned to the signal points are determined from a *partition* of the constellation. A 2^ν-ary modulation signal set $\mathcal{S}$ is *partitioned* in ν levels. For $1 \leq i \leq \nu$, at the i-th partition level, the signal set is divided into two subsets $\mathcal{S}_i(0)$ and $\mathcal{S}_i(1)$, such that the *intra-set distance*, δ_i^2, is maximized. A *label bit* $b_i \in \{0, 1\}$ is associated with the subset choice, $\mathcal{S}_i(b_i)$, at the i-th partition level. This partitioning process results in a *labeling* of the signal points. Each signal point in the set has a unique ν-bit label $b_1 b_2 \cdots b_\nu$, and is denoted by $s(b_1, b_2, \cdots, b_\nu)$. With this *standard (Ungerboeck) partitioning* of a 2^ν-ary modulation signal constellation, the intra-set distances are in nondecreasing order $\delta_1^2 \leq \delta_2^2 \leq \cdots \leq \delta_\nu^2$. This strategy corresponds to a *natural labeling* for M-PSK modulations, i.e., binary representations of integers, whose

value increases clockwise (or counter-wise). Figure 103 shows a natural mapping of bits to signals for the case of 8-PSK modulation, with $\delta_1^2 = 0.586$, $\delta_2^2 = 2$, and $\delta_3^2 = 4$.

Ungerboeck regarded the encoder *"simply as a finite-state machine with a given number of states and specified state transitions"*. He gave a set of pragmatic rules to map signal subsets and points to branches in a trellis. These rules can be summarized as follows:

Rule 1:

All subsets should occur in the trellis with equal frequency.

Rule 2:

State transitions that begin and end in the same state should be assigned subsets separated by the largest Euclidean distance.

Rule 3:

Parallel transitions are assigned signal points separated by the largest Euclidean distance (the highest partition levels).

The general structure of a TCM encoder is shown in Figure 104. In the general case of a rate $(\nu - 1)/\nu$ TCM system, the trellis structure is inherited from a $k/(k+1)$ convolutional encoder. The uncoded bits introduce *parallel branches* in the trellis.

Example 96 In this example, a 4-state rate-2/3 TCM system is considered. A constellation for 8-PSK modulation is shown in Figure 103. The spectral efficiency is $\mu = 2$ bits/symbol. A block diagram of the encoder is shown in Figure 105. The binary convolutional code is the same memory-2 rate-1/2 code that was used in Chapter 5. Note from Figure 106 that the trellis structure is the same as that of the binary convolutional code, with the exception that every branch in the original diagram is replaced by two parallel branches, associated with the uncoded bit u_1.

9.2.2 *Maximum-likelihood decoding*

The Viterbi algorithm[2] can be applied to decode the most likely TCM sequences, provided that the branch metric generator is modified to include parallel branches. Also, the selection of the winning branch and surviving uncoded bits should be changed. The survivor path (or trace-back) memory should include the $(\nu - 1 - k)$ uncoded bits, as opposed to just one bit for rate-$1/n$ binary convolutional codes. It is also important to note that in 2^ν-ary PSK or QAM modulation, the *correlation metrics* for *two-dimensional* symbols are of the form $x_p x_r + y_p y_r$, where (x_p, y_p) is a reference signal point in the constellation and (x_r, y_r) is the received signal point. All other implementation issues discussed in Sections 5.4 and 7.2 apply to TCM decoders.

9.2.3 *Distance considerations and error performance*

The error performance of TCM can be analysed in the same way as for convolutional codes. That is, a weight enumerating sequence can be obtained from the state diagram of the TCM

[2] The Viterbi algorithm is discussed in Sections 5.4 and 7.2.

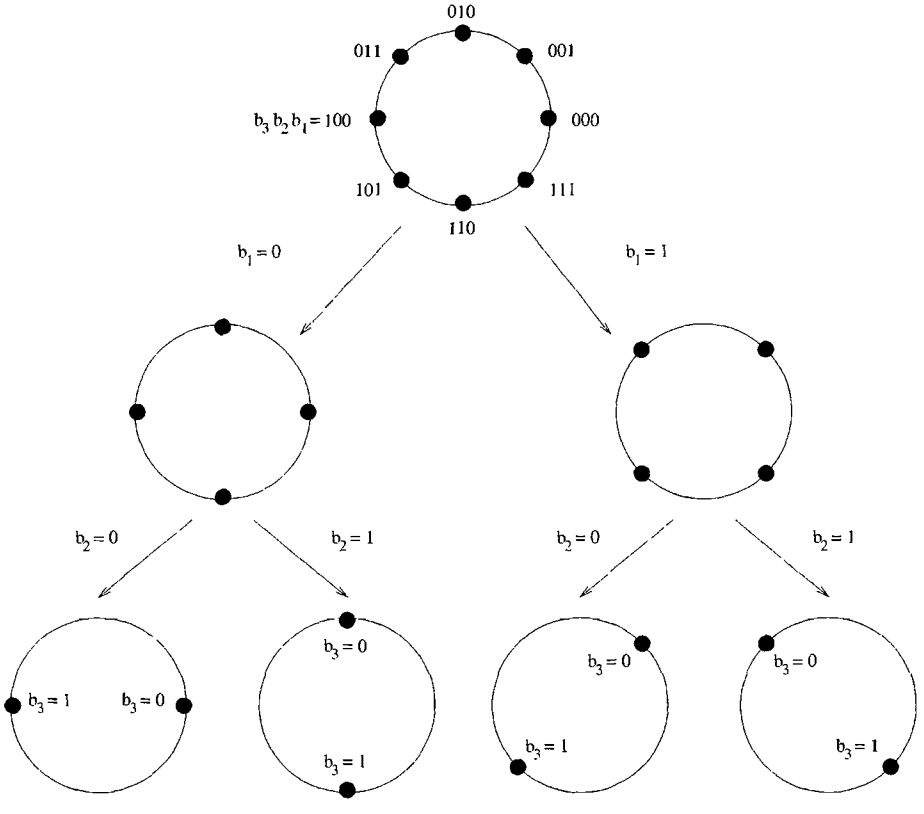


Figure 103 Natural mapping of an 8PSK constellation.

encoder, as in Section 5.3. The only difference is that the powers are not integers (Hamming distances) but real numbers (Euclidean distances). Care needs to be taken of the fact that the state transitions contain parallel branches. This means that the labels of the modified state diagram contain two terms. See [BDMS].

Example 97 Figure 107 shows the modified state diagram for the 4-state TC 8-PSM modulation of Example 96. The branches in the trellis have been labeled with integers corresponding to the eight phases of the modulation signals. To compute the weight enumerating sequence $T(x)$, the same procedure as in Section 5.3 is applied. Alternatively, by directly analysing the trellis structure in Figure 108, it can be deduced that the MSED between coded sequences is

$$D_C^2 = \min\{D_{\text{par}}^2, D_{\text{tre}}^2\} = 3.172,$$

which, when compared to an uncoded QPSK modulation system with the same spectral efficiency of 2 bits/symbol, gives an asymptotic coding gain of 2 dB.

9.2.4 Pragmatic TCM and two-stage decoding

For practical considerations, it was suggested in [Vit4, ZW] that the 2^ν-ary modulation signal constellations be partitioned in such a way that the cosets at the top two partition levels are

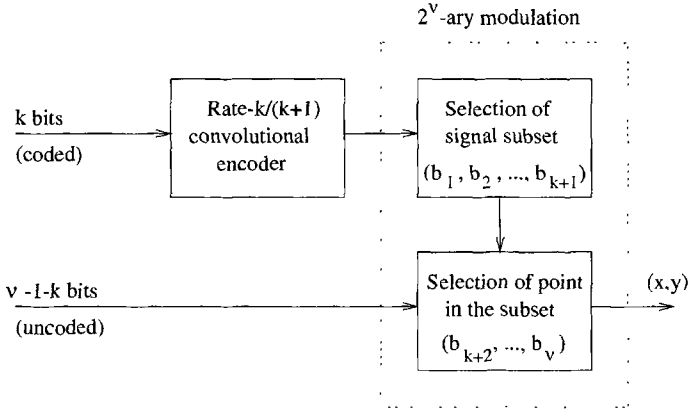


Figure 104 General encoder of rate-$(\nu - 1)/\nu$ trellis-coded modulation.

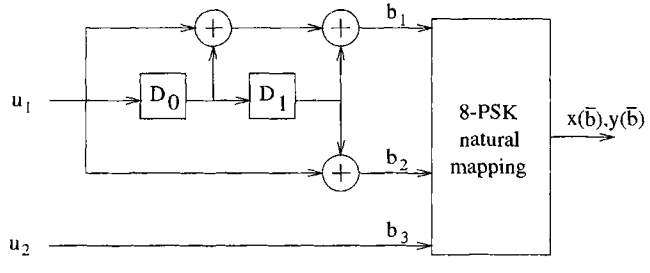


Figure 105 Encoder of a 4-state rate-2/3 trellis coded 8-PSK modulation.

associated with the outputs of the standard memory-6 rate-1/2 convolutional encoder. This mapping leads to a *pragmatic TCM system*. With respect to the general encoder structure in Figure 104, the value of $k = 1$ is fixed, as shown in Figure 109. As a result, the trellis structure of pragmatic TCM remains the same, as opposed to Ungerboeck-type TCM, for all values of $\nu > 2$. The difference is that the number of parallel branches $\nu - 2$ increases with the number of bits per symbol. This suggests a two-stage decoding method in which, at the first stage, the parallel branches in the trellis "collapse" into a single branch, and a conventional off-the-shelf Viterbi decoder used to estimate the coded bits associated with the two top partition levels. In a second decoding stage, based on the estimated coded bits and the positions of the received symbols, the uncoded bits are estimated. Figure 110 is a block diagram of a two-stage decoder of pragmatic TCM.

In [MM], a symbol transformation is applied to the incoming symbols that enables use of a Viterbi decoder without changes in the branch metric computation stage. The decoding procedure is similar to that presented in [CRKO, PS], with the exception that, with symbol transformation, the Viterbi algorithm can be applied as if the signals were BPSK (or QPSK) modulated. This method is described below for M-PSK modulation.

Specifically, let (x, y) denote the I and Q coordinates of a received M-PSK symbol with amplitude $r = \sqrt{x^2 + y^2}$ and phase $\phi = \tan^{-1}(y/x)$. Based on ϕ, a transformation is applied such that the M-PSK points are mapped into "coset" points labeled by the outputs of a rate-1/2 64-state convolutional encoder.

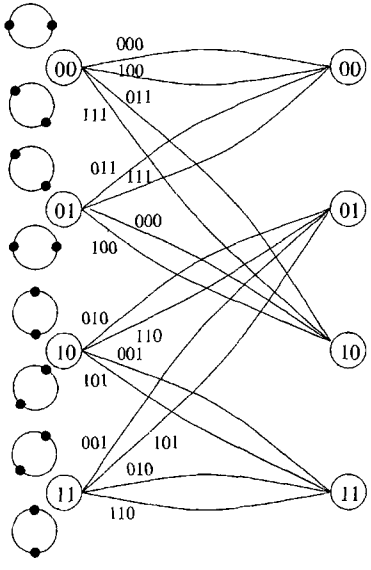


Figure 106 Trellis structure of a rate-2/3 trellis coded 8-PSK modulation based on the encoder of Figure 105.

For TCM with M-ary PSK modulation, $M = 2^\nu$, $\nu > 2$, let ξ denote the number of *coded bits per symbol*[3], where $\xi = 1, 2$. Then the following rotational transformation is applied to each received symbol, (x, y), to obtain an input symbol (x', y') to the VD.

$$\begin{aligned} x' &= r\cos\left[2^{\nu-\xi}(\phi - \Phi)\right], \\ y' &= r\sin\left[2^{\nu-\xi}(\phi - \Phi)\right], \end{aligned} \tag{9.5}$$

where Φ is a constant phase rotation of the constellation that affects all points equally. Under the transformation (9.5), a $2^{m-\xi}$-PSK coset in the original 2^m-PSK constellation "collapses" into a coset point in a 2^ξ-PSK coset constellation in the $x' - y'$ plane.

Example 98 A rate-2/3 trellis-coded 8-PSK modulation with 2 coded bits per symbol is considered. Two information bits (u_1, u_2) are encoded to produce three coded bits (u_2, v_2, v_1), which are mapped onto an 8-PSK signal point, where (v_2, v_1) are the outputs of the standard rate-1/2 64-state convolutional encoder[4]. The signal points are labeled by bits (u_2, v_2, v_1), and the pair (v_2, v_1) is the index of a coset of a BPSK subset in the 8-PSK constellation, as shown at the top of Figure 111.

In this case $\phi' = 2\phi$ and, under the rotational transformation, a BPSK subset in the original 8-PSK constellation collapses to a coset point of the QPSK coset constellation in the $x' - y'$ plane, as shown in Figure 111. Note that both points of a given BPSK coset have the same value of ϕ'. This is because their phases are given by ϕ and $\phi + \pi$.

---

[3] The case $\xi = 2$ corresponds to conventional TCM with 8-PSK modulation. The case $\xi = 1$ is used in TCM with coded bits *distributed over two 8-PSK signals*, such as the rate-5/6 8-PSK modulation scheme proposed in the DVB-DSNG specification [DVB].

[4] v_1 and v_2 are the outputs from generators 171 and 133, in octal, respectively.

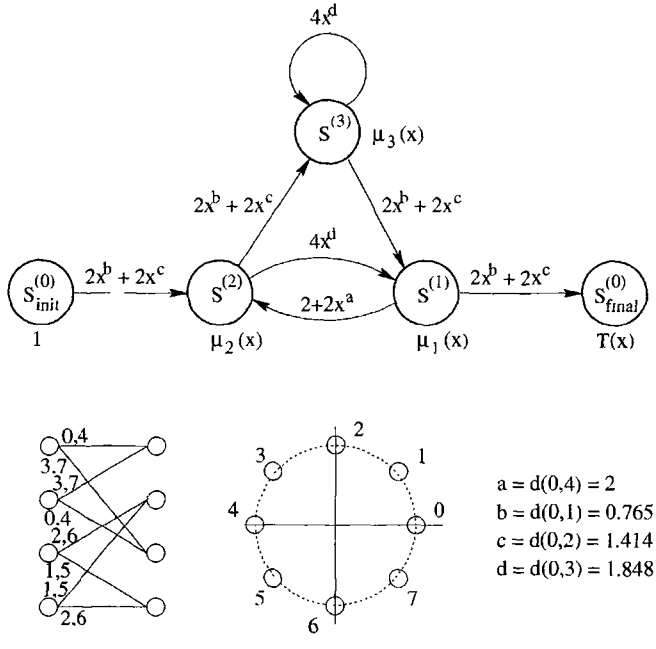


Figure 107 *The modified state diagram of a 4-state TC 8-PSK modulation scheme.*

The output of the VD is an estimate of the *coded* information bit, u_1. In order to estimate the *uncoded* information bit, u_2, it is necessary to re-encode u_1 to determine the most likely coset index. This index and a *sector* in which the received 8-PSK symbol lies can be used to decode u_2. For a given coset, each sector S gives the closest point (indexed by u_2) in the BPSK pair to the received 8-PSK symbol. For example, if the decoded coset is (1,1) and the received symbol lies within sector 3, then $u_2 = 0$, as can be verified from Figure 111.

Figure 112 shows simulation results of MLD (denoted with the legend "TC8PSK_23_SSD") and two-stage decoding of pragmatic TC-8PSK (legend "TC8PSK_23_TSD"). With two-stage decoding, a loss in performance of only 0.2 dB is observed compared with MLD.

A similar transformation can be applied in the case of M-QAM, the difference is that the transformation is based solely on the I-channel and Q-channel symbols. That is, there is no need to compute the phase. An example is shown in Figure 113 for TC 16-QAM with $\xi = 2$ coded bits per symbol. The coded bits are now the indexes of cosets of QPSK subsets. The transformation of 16-QAM modulation ($\xi = 2$) is given by a kind of "modulo 4" operation:

$$x' = \begin{cases} x, & |x| \leq 2; \\ (x-4), & x \geq 2; \\ (x+4), & x < 2, \end{cases}$$

$$y' = \begin{cases} y, & |y| \leq 2; \\ (y-4), & y \geq 2; \\ (y+4), & y < 2. \end{cases}$$

Finally, it is interesting to note that a pragmatic TCM system with a turbo code as component code was recently proposed in [WM].

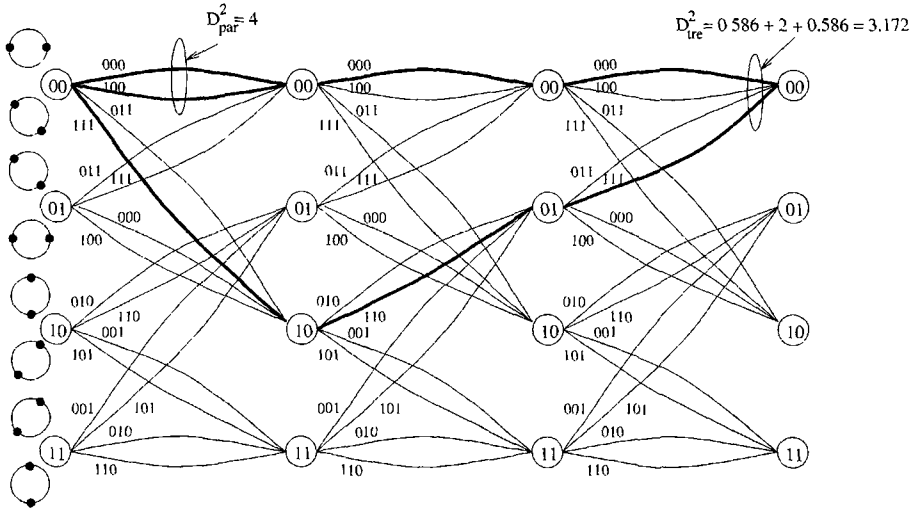


Figure 108 Two paths at minimum squared Euclidean distance in the trellis of Example 96.

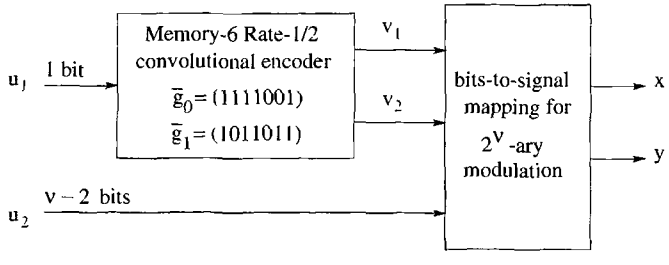


Figure 109 Block diagram of an encoder of pragmatic TCM.

9.3 Multilevel coded modulation (MCM)

In the Imai-Hirakawa multilevel coding scheme [IH], the 2^ν-ary modulation signal set is binary partitioned in ν levels. The components of codewords of ν binary component codes C_i, $1 \leq i \leq \nu$, are used to index the cosets at each partition level. One of the advantages of MCM is the flexibility of designing coded modulation schemes by coordinating the intra-set Euclidean distances, δ_i^2, $i = 1, 2, \cdots, \nu$, at each level of set partitioning, and the minimum Hamming distances of the component codes. Wachsmann et al. [WFH] have proposed several design rules that are based on capacity (by applying the chain rule of mutual information) arguments. Moreover, multilevel codes with long binary component codes, such as turbo codes or LDPC codes, were shown to achieve capacity [WFH, For8].

It also worthwhile noting that, while generally binary codes are chosen as component codes, i.e., the partition is binary, in general the component codes can be chosen from any finite field $GF(q)$ matching the partition of the signal set. Another important advantage of multilevel coding is that (binary) decoding can be performed separately at each level. This *multistage decoding* results in greatly reduced complexity, compared with MLD for the overall code.

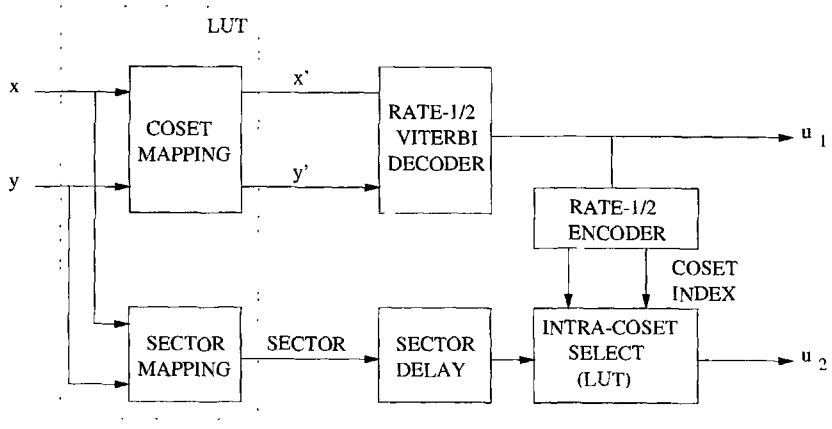


Figure 110 Block diagram of a two-stage decoder of pragmatic TCM.

9.3.1 Constructions and multi-stage decoding

For $1 \leq i \leq \nu$, let C_i denote a binary linear block (n, k_i, d_i) code. Let $\bar{v}_i = (v_{i1}, v_{i2}, \cdots, v_{in})$ be a codeword in C_i, $1 \leq i \leq \nu$. Consider a *permuted time-sharing code* $\pi(|C_1|C_2|\cdots|C_\nu|)$, with codewords

$$\bar{v} = (\, v_{11}v_{21}\cdots v_{\nu 1} \quad v_{12}v_{22}\cdots v_{\nu 2} \quad \cdots \quad v_{1n}v_{2n}\cdots v_{\nu n}\,).$$

Each ν-bit component in $\bar{v}$ is the *label* of a signal in a 2^ν-ary modulation signal set $\mathcal{S}$. Then

$$s(\bar{v}) = (s(v_{11}v_{21}\cdots v_{\nu 1}), s(v_{12}v_{22}\cdots v_{\nu 2}), \cdots, s(v_{1n}v_{2n}\cdots v_{\nu n})$$

is a sequence of signal points in $\mathcal{S}$.

The following collection of signal sequences over $\mathcal{S}$,

$$\Lambda \triangleq \{s(\bar{v}) : \bar{v} \in \pi(|C_1|C_2|\cdots|C_\nu|)\},$$

forms *a ν-level modulation code over the signal set $\mathcal{S}$*, or a ν-level coded 2^ν-ary modulation. The same definition can be applied to convolutional component codes.

The rate, or spectral efficiency, of this coded modulation system, in bits/symbol, is $R = (k_1 + k_2 + \cdots + k_\nu)/n$. The MSED of this system, denoted by $D^2_C(\Lambda)$, is given by [IH]

$$D^2_C(\Lambda) \geq \min_{1 \leq i \leq \nu} \{d_i \delta_i^2\}. \tag{9.6}$$

Example 99 In this example, a three-level block coded 8-PSK modulation system is considered. The encoder structure is depicted in Figure 114. Assuming a unit-energy 8-PSK signal set, and with reference to Figure 103, note that the MSED at each partition level are $\delta_1^2 = 0.586$, $\delta_2^2 = 2$ and $\delta_3^2 = 4$.

The MSED of this coded 8-PSK modulation system is:

$$D^2_C(\Lambda) = \min\{d_1\delta_1^2, d_2\delta_2^2, d_3\delta_3^2\} = \min\{8 \times 0.586, 2 \times 2, 1 \times 4\} = 4,$$

and the coding gain is 3 dB with respect to uncoded QPSK The trellises of the component codes are shown in Figure 115. The overall trellis is shown in Figure 116.

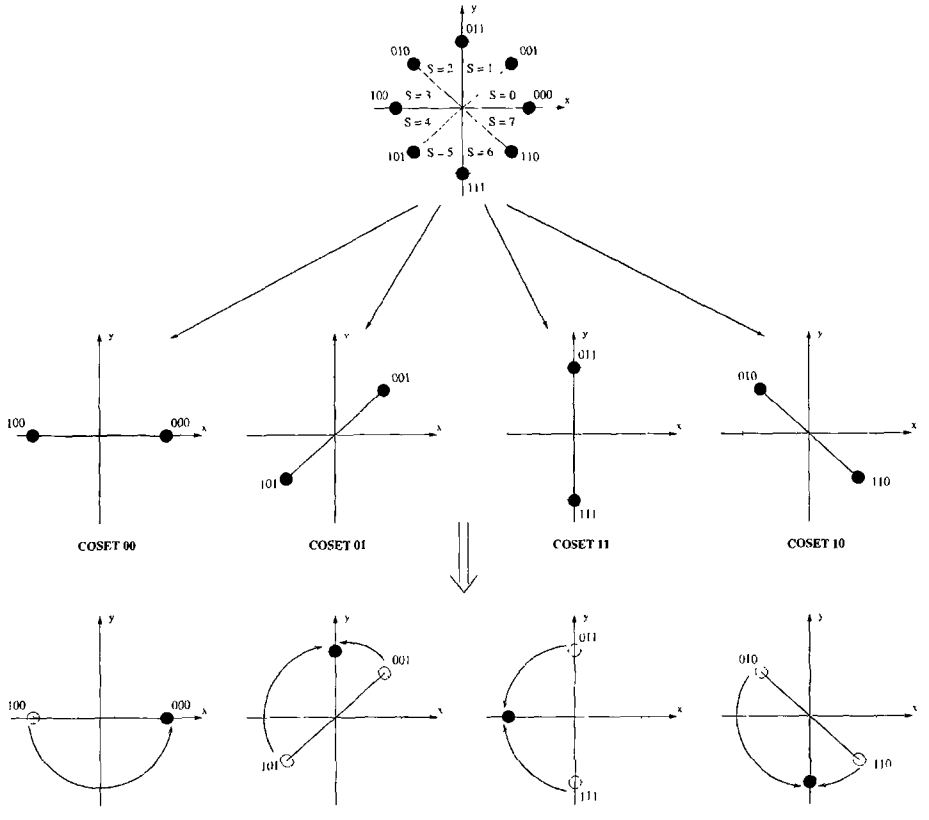


Figure 111 Partitioning of an 8-PSK constellation ($\xi = 2$) and coset points.

As mentioned before, one of the advantages of multilevel coding is that multistage decoding can be applied. Figures 117 (a) and (b) show the basic structures used in encoding and decoding of multilevel codes. Multistage decoding results in reduced complexity (e.g., measured as number of branches in trellis decoding), compared to MLD decoding (e.g., using the Viterbi algorithm and the overall trellis.) However, in multistage decoding, the decoders at early levels regard the later levels as uncoded. This results in more codewords at minimum distance, i.e., an increase in *error multiplicity* or *number of nearest neighbors*. The value of this loss depends on the choice of the component codes and the bits-to-signal mapping, and for BER $10^{-2} \sim 10^{-5}$ can be in the order of several dB.

Example 100 In this example, multistage decoding of three-level coded 8-PSK modulation is considered. The decoder in the first stage uses the trellis of the first component code C_1. Branch metrics are the distances (correlations) from the subsets selected at the first partitioning level to the received signal sequence, as illustrated in Figure 118.

Once a decision is made in the first stage, it is passed on to the second stage. The decoder in the second stage uses the trellis of the second component code with information from the first stage. For 8-PSK modulation, if the decoded bit in the first stage is $b_1 = 0$, then the received signal sequence is unchanged. If the decoded bit is $b_1 = 1$, then the received signal is rotated by 45°. Again, branch metrics are distances (correlations) from the subsets selected at the

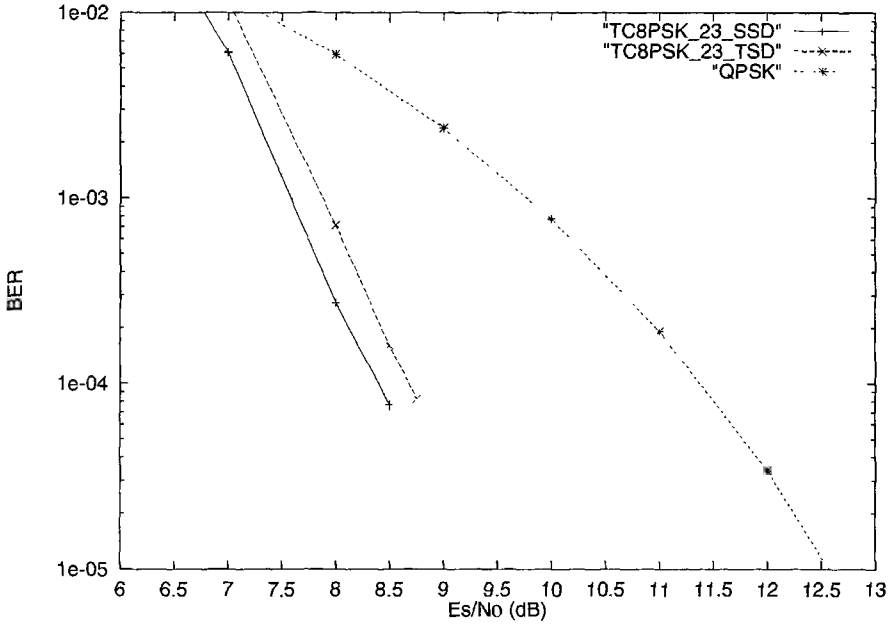


Figure 112 Simulation results of MLD versus two-stage decoding for pragmatic 8-PSK modulation.

second partitioning stage — given the decision at the first decoding stage — to the receive signal sequence.

Finally, based on the decisions in the first two decoding stages, the decoder of the third component is used. The branch metrics are the same as for BPSK modulation. There are four rotated versions of the BPSK constellation, in accordance with the decisions in the first two decoding stages. Therefore one approach is to rotate the received signal according to the decisions on $b_1 b_2$ and use the same reference BPSK constellation. This is illustrated in Figure 119.

For medium to large code lengths, hybrid approaches may be the way to go for ultimate MCM performance, with powerful turbo codes used in the top partition levels and binary codes with hard-decision decoding assigned to lower partition levels. These combinations can achieve excellent performance [WFH].

9.3.2 Unequal-error-protection with MCM

Because of its flexibility in designing the minimum Euclidean distances between coded sequences at each partition level, MCM is an attractive scheme to achieve *unequal error protection* (UEP). However, great care has to be exercised in choosing the bits-to-signal mapping, so that the desired UEP capabilities are not destroyed. This issue was investigated in [MFLI, IFMLI], where several partitioning approaches were introduced that constitute generalizations of the block [WFH] and Ungerboeck [Ung1] partitioning rules.

In these *hybrid partitioning* approaches, some partition levels are nonstandard while at other levels partitioning is performed using Ungerboeck's rules [Ung1]. In this manner, a

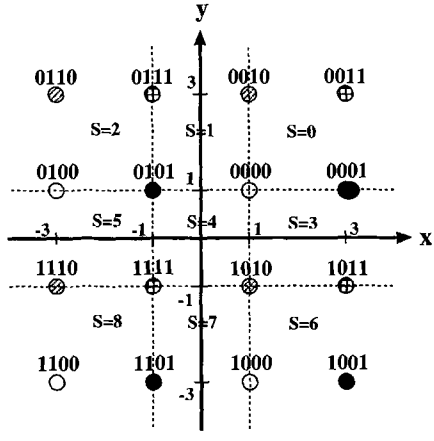


Figure 113 16-QAM constellation for pragmatic TCM with two-stage decoding.

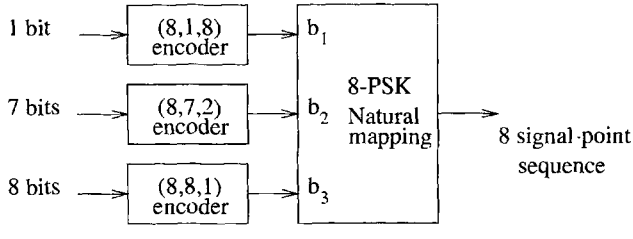


Figure 114 Example of MCM with 8-PSK modulation.

good tradeoff is obtained between error coefficients and intra-level Euclidean distances. To achieve UEP capabilities, the Euclidean distances at each partition level are chosen such that

$$d_1\delta_1^2 \geq d_2\delta_2^2 \geq \cdots \geq d_\nu\delta_\nu^2. \tag{9.7}$$

For $1 \leq i \leq \nu$, let $\bar{v}_i(u_i)$ be the codeword of C_i in correspondence to a k_i-bit message vector $\bar{u}_i$, and let $\bar{s} = \bar{s}(\bar{u})$ and $\bar{s}' = \bar{s}(\bar{u}')$ denote coded 2^ν-ary modulation signal sequences corresponding to message vectors $\bar{u} = (\bar{u}_1, \bar{u}_2, \cdots, \bar{u}_\nu)$ and $\bar{u}' = (\bar{u}'_1, \bar{u}'_2, \cdots, \bar{u}'_\nu)$, respectively. The *Euclidean separations* [YI] between coded sequences at the i-th partition level, for $i = 1, \cdots, \nu$, are defined as

$$\mathbf{s}_i \triangleq \min\left\{d(\bar{s}, \bar{s}') : \bar{u}_i \neq \bar{u}'_i, \bar{u}_j = \bar{u}'_j, j < i\right\}, \tag{9.8}$$

with $\mathbf{s}_1 = d_1\delta_1^2$, $\mathbf{s}_2 = d_2\delta_2^2$, $\cdots$, $\mathbf{s}_\nu = d_\nu\delta_\nu^2$. For transmission over an AWGN channel, the set of inequalities (9.7) results in message vectors with decreasing error protection levels.

It is known from [WFH] that Ungerboeck's partitioning rules [Ung1] are inappropriate for multistage decoding of multilevel coded modulations, at low to medium signal-to-noise ratios, because of the large number of nearest neighbor sequences (NN) in the first decoding stages.

Example 101 Figure 120 shows simulation results of the performance of a three-level coded 8-PSK modulation with the $(64, 18, 22)$, $(64, 57, 4)$ and $(64, 63, 2)$ extended BCH codes (ex-BCH codes) as component codes C_i, $i = 1, 2, 3$, respectively. The Euclidean separations are

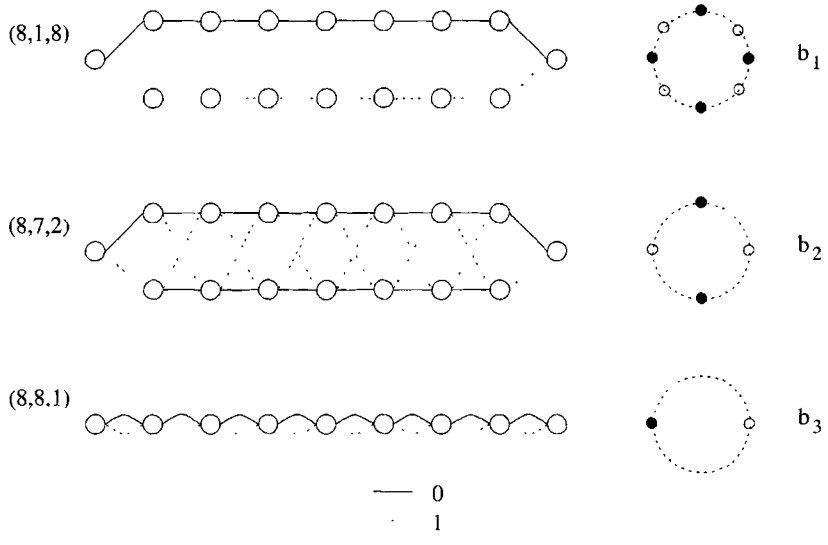


Figure 115 Trellises of component codes of an example MCM with 8-PSK modulation.

$s_1 = 12.9$, $s_2 = s_3 = 8$, for 18 and 120 information bits, respectively (asymptotic coding gains of 8.1 dB and 6 dB, respectively). The adverse effects of the number of NN (or error coefficient) in the first decoding stage are such that the coding gains are greatly reduced.

In the following, a UEP scheme based on nonstandard partitioning is presented. The reader is referred to [WFH, MFLI, IFMLI] for details on multilevel coding design for both conventional (equal error protection) and UEP schemes.

Nonstandard partitioning

The block partitioning [WFH] shown in Figure 121 (a) is used to construct three-level coded 8-PSK modulation schemes with UEP. In the figure, the color black is used to represent signal points whose label is of the form $\mathbf{0}b_2b_3$, with $b_2, b_3 \in \{0, 1\}$. Similarly, the color white is used for points with labels $\mathbf{1}b_2b_3$. A circle indicates that the label is of the form $b_1\mathbf{0}b_3$, $b_1, b_3 \in \{0, 1\}$, while a square is used to represent signal points with labels $b_1\mathbf{1}b_3$.

It can be seen from Figure 121 (b) that in order to determine the value of the first label bit, b_1, only the X-coordinate is sufficient. If a signal point is on the left-hand half plane ($X < 0$) then it corresponds to $b_1 = 0$, otherwise it corresponds to $b_1 = 1$. In the same way, the Y-coordinate suffices to determine the value of the second label bit b_2. If a signal point lies in the upper half plane ($Y > 0$), then $b_2 = 0$, otherwise $b_2 = 1$. This property of block partitioning allows the first and second levels to be decoded *independently* or *in parallel*. A similar observation led to the development of *parallel decoding* (PD) for multilevel codes with Gray mapping in [Schr].

Multistage decoding

In the first and second decoding stages, the decision variable is just the projection of the received signal sequence onto the X or Y axis, respectively. Figure 121 (c) shows a block

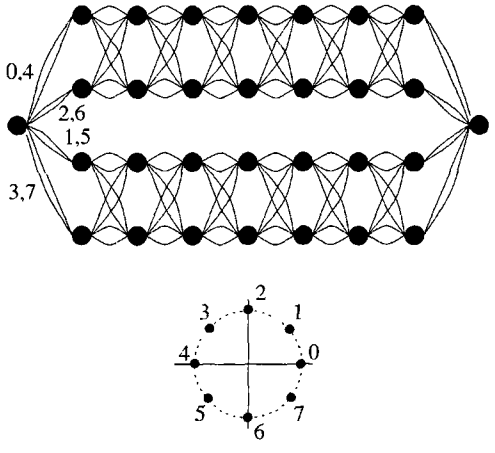


Figure 116 Overall trellis of an example MCM with 8-PSK modulation.

diagram of a multistage decoder for a three-level coded 8-PSK modulation with block partitioning. The decoders for the first and second stages operate independently on the in-phase and quadrature component of the received signal sequences, $\bar{r}_x$ and $\bar{r}_y$, respectively. Once decisions are made as to the estimates of the corresponding codewords, $\hat{v}_1$ and $\hat{v}_2$, they are passed on to the third decoding stage.

Let $\hat{v}_i = (\hat{v}_{i1}, \hat{v}_{i2}, \cdots, \hat{v}_{in}) \in C_i$ be the decoded codeword at the i-th stage, $i = 1, 2$. Before the third-stage decoding, each two-dimensional coordinate (r_{xj}, r_{yj}) of the received signal $\bar{r} = (\bar{r}_x, \bar{r}_y)$ is projected onto a one dimensional coordinate r'_{xj}, $1 \leq j \leq n$. The values r'_{xj} are the decision variables used by the decoder of C_3. The projection depends on the decoded quadrant, which is indexed by the pair $(\hat{v}_{1j}, \hat{v}_{2j})$, $1 \leq j \leq n$, as shown in the table below.

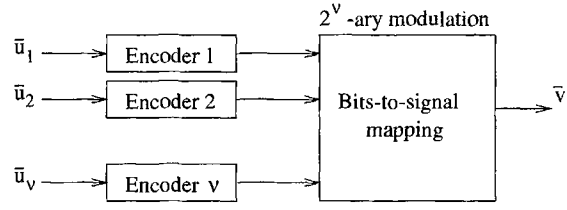
$\hat{v}_{1j}$	$\hat{v}_{2j}$	r'_{xj}
0	0	$-\sqrt{2}/2\,(r_{xi} - r_{yi})$
0	1	$-\sqrt{2}/2\,(r_{xi} + r_{yi})$
1	1	$\sqrt{2}/2\,(r_{xi} - r_{yi})$
1	0	$\sqrt{2}/2\,(r_{xi} + r_{yi})$

This is a scaled rotation of $\bar{r}$ by $\pi/4$, so that the *rotated sequence* $\bar{r}' = (r'_{x1}, r'_{x2}, \cdots, r'_{xn})$ can be decoded using a soft-decision procedure for component code C_3. Note that, unlike Ungerboeck partitioning, the independence between the first and second levels in block partitioning results in *no error propagation* from the first decoding stage to the second.

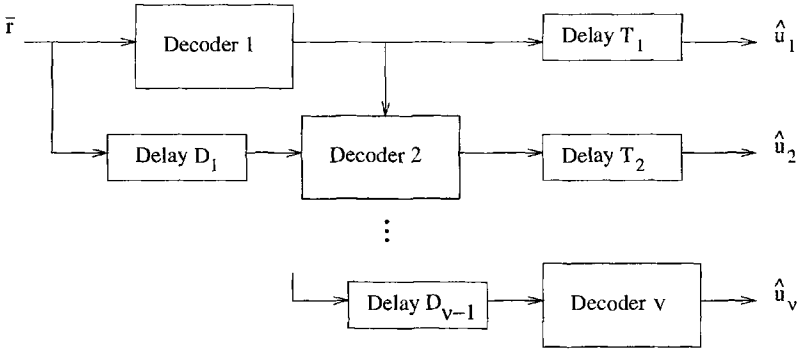
For $i = 1, 2, \cdots, \nu$, let $A_w^{(i)}$ denote the number of codewords in C_i of weight w. Assuming systematic encoding, a union bound on the bit error probability of the first decoding stage can be written as [MFLI, IFMLI]

$$P_{b1}^{(NS)} \leq \sum_{w=d_1}^{n} \frac{w}{n} A_w^{(1)} 2^{-w} \sum_{i=0}^{w} \binom{w}{i} Q\left(\sqrt{\frac{2RE_b}{N_0} d_P^2(i)}\right), \tag{9.9}$$

where $d_P^2(i) = \frac{1}{w}\left[i\Delta_1 + (w-i)\Delta_2\right]^2$. The probability of a bit error in the second decoding



(a) Multilevel Encoder



(b) Multistage Decoder

Figure 117 Basic structures of an encoder and a decoder of multi-level coded modulation systems.

stage upper is also bounded by (9.9) using the same arguments above.

The bound (9.9) can be compared with a similar one for the Ungerboeck's partitioning (UG) strategy:

$$P_{b1}^{(UG)} \leq \sum_{w=d_1}^{n} \frac{w}{n} A_w^{(1)} 2^w Q\left(\sqrt{\frac{2R\overline{E}_b}{N_0} w\Delta_1^2}\right). \tag{9.10}$$

From (9.9) and (9.10), it is observed that, while Ungerboeck's partitioning increases exponentially the effect of nearest neighbor sequences, by a factor of 2^w, the block partitioning has for $d_P^2(w) = w\Delta_1^2$ an error coefficient term, 2^{-w}, that *decreases exponentially* with the distances of the first-level component code. As a result, for practical values of E_b/N_O, the block partitioning may yield, at the first stage, a real coding gain *even greater than the asymptotic coding gain*. This is a very desirable feature of a coded modulation with UEP.

For nonstandard partitioning (NS), the second level is generally designed to have a larger coding gain than the third level. Under this assumption, a good approximation is obtained by assuming that decoding decisions in the first and the second decoding stages are correct,

$$P_{b3}^{(NS)} \lesssim \sum_{w=d_3}^{n} \frac{w}{n} A_w^{(3)} Q\left(\sqrt{\frac{2R\overline{E}_b}{N_0} w\Delta_1^2}\right). \tag{9.11}$$

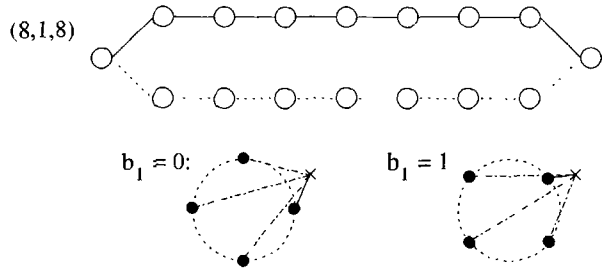


Figure 118 Trellis and symbols used in metric computations in the first decoding stage.

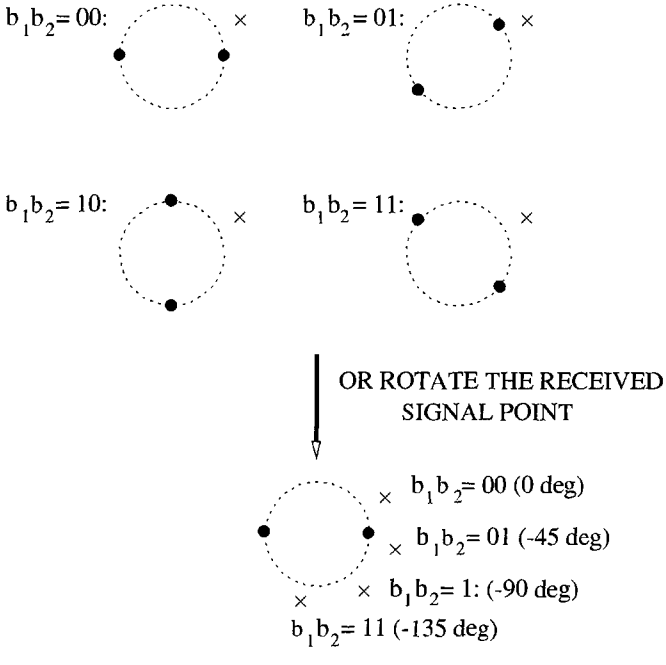


Figure 119 Trellis and symbols used in metric computations in the third decoding stage.

Example 102 Consider a three-level 8-PSK modulation for UEP with extended BCH $(64, 18, 22)$, $(64, 45, 8)$ and $(64, 63, 2)$ codes as the first-, second- and third-level codes, respectively. This coding scheme has rate equal to 1.97 bits per symbol and can be compared with uncoded QPSK modulation, which has approximately the same rate (a difference of only 0.06 dB). Simulation results are shown in Figure 122. S1(n, k) and UB(n, k) denote simulations and upper bounds. An large coding gain of 8.5 dB is achieved at the BER of 10^{-5} for 18 most important bits (14.3%) encoded in the first level. In the second and third stages, the corresponding values of coding gain are 2.5 dB and -4.0 dB, respectively[5].

[5] Note that at this BER, the simulated coding gain at the first decoding stage is even greater than the asymptotic coding gain (8.1 dB), because of the reduced error coefficients.

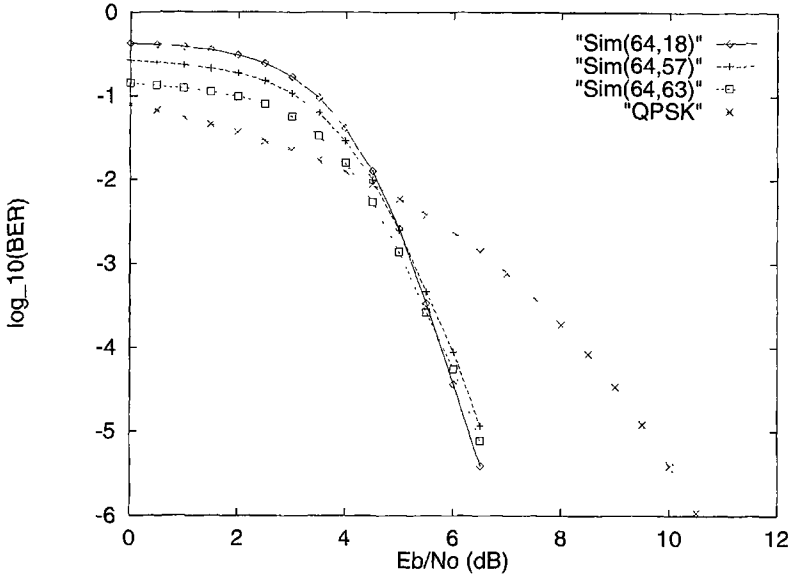


Figure 120 Simulation results of a three-level coded 8-PSK modulation with Ungerboeck mapping.

9.4 Bit-interleaved coded modulation (BICM)

In [CTB1, CTB2], the ultimate approach to pragmatic coded modulation is presented. The system consists of binary encoding followed by a pseudo-random *bit interleaver*. The output of the interleaver is grouped in blocks of ν bits which are assigned, via a *Gray mapping*, to points in a 2^ν-ary modulation constellation. The capacity of this *bit-interleaved coded modulation* (BICM) scheme has been shown to be surprisingly close to the capacity of TCM, when Gray mapping is employed. Moreover, over flat Rayleigh fading channels, BICM outperforms a CM with symbol interleaving [CTB2]. A block diagram of a BICM system is shown in Figure 123.

Let $\mathcal{X}$ denote a 2^ν-ary signal constellation with minimum distance $D_{\min}$. For $i = 1, 2, \cdots, \nu$, let $\ell^i(x)$ be the i-th bit of the label of a signal point x, and let $\mathcal{X}_b^i \subset \mathcal{X}$ the subset of signal points with labels such that the i-th bit has a value $b \in \{0, 1\}$.

9.4.1 Gray mapping

A one-to-one and onto binary map m from $\{0,1\}^\nu$ to $\mathcal{X}$ is a Gray mapping if, for all $i = 1, 2, \cdots, \nu$, and $b \in \{0, 1\}$, each $x \in \mathcal{X}_b^i$ has at most one nearest neighbor $y \in \mathcal{X}_{b'}^i$ at distance $D_{\min}$, where $b' = b \oplus 1$. Gray mapping is the key component of a BICM system. Its main function is – ideally – to produce an equivalent channel that has ν parallel, independent and memoryless, binary channels. Each channel corresponds to a position in the label of a signal $x \in \mathcal{X}$. For each codeword at the output of the binary encoder, the interleaver assigns at random a position in the label of the signals to transmit the coded bits.

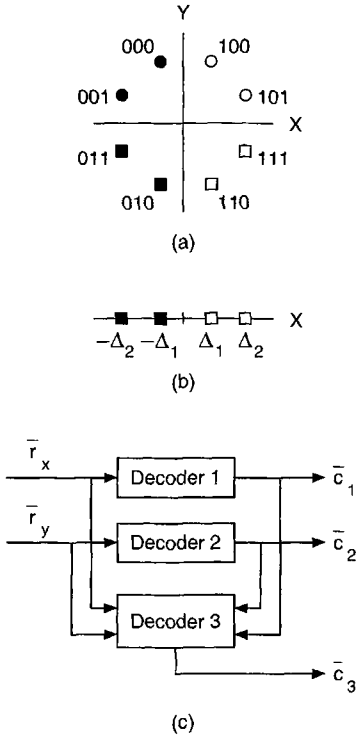


Figure 121 An 8-PSK constellation with block partitioning: (a) labeling; (b) X coordinate projections; (c) decoder structure. ($\bar{c}_i$ denote estimated codewords in C_i, $i = 1, 2, 3$.)

9.4.2 Metric generation: De-mapping

Before the description of how metrics for an MLD decoder are generated, some notation is needed. Let r denote the channel output after transmission of x. Assuming uniform input distribution, the conditional probability of r given $\ell^i(x) = b$ is

$$p(r|\ell^i(x) = b) = \sum_{x \in \mathcal{X}} p(r|x)p(x|\ell^i(x) = b) = 2^{-(\nu-1)} \sum_{x \in \mathcal{X}_b^i} p(r|x). \tag{9.12}$$

Let i denote the position of the coded bit v_j in the label of $x_{\pi(j)}$. At each time j, let v_j be a code symbol and $x_{\pi(j)}$ the interleaved signal point, received as $r_{\pi(j)}$ after transmission over a noisy channel.

The receiver then produces *bit metrics*

$$\lambda^i(r_{\pi(j)}, b) = \log \left(\sum_{x \in \mathcal{X}_b^i} p(r_{\pi(j)}|x) \right), \tag{9.13}$$

for $b = 0, 1$ and $i = 1, 2, \cdots, \nu$.

An MLD algorithm, such as the Viterbi algorithm, uses the above metrics and makes decisions based on the rule

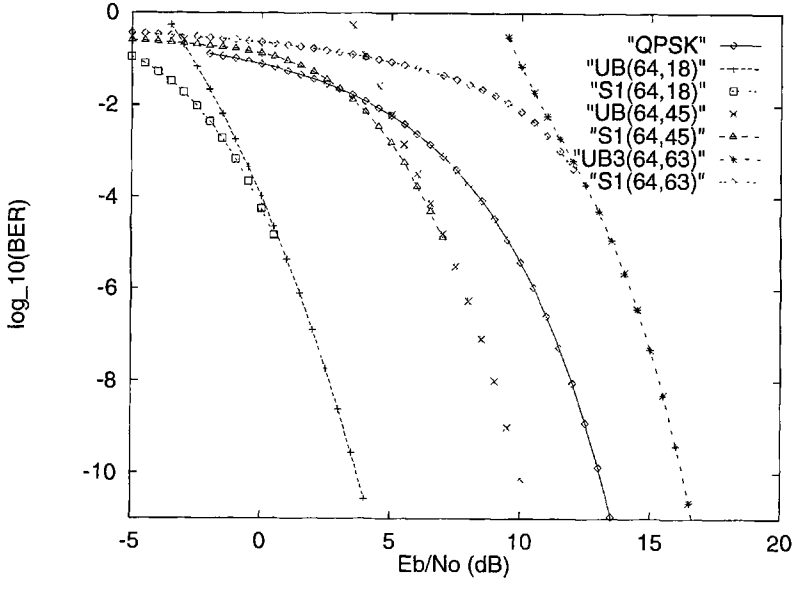


Figure 122 Simulation results of a three-level coded 8-PSK modulation with UEP capability. BCH component codes and block partitioning.

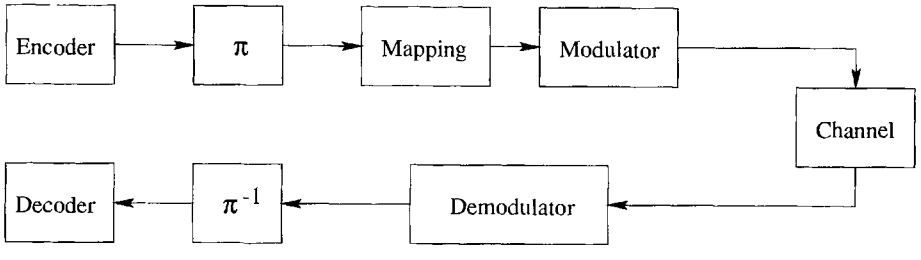


Figure 123 A bit-interleaved coded modulation system.

$$\hat{v}_i = \arg\max_{\bar{v}\in C} \sum_j \lambda^i(r_{\pi(j)}, v_j). \tag{9.14}$$

As before, a max-log approximation of (9.13) is possible, resulting in the approximated bit metric,

$$\lambda_a^i(r_{\pi(j)}, b) = \max_{x\in\mathcal{X}_b^i} \log p(r_{\pi(j)}|x). \tag{9.15}$$

9.4.3 Interleaving

With transmission over an AWGN channel, a short interleaver will suffice. The main purpose is to break the correlation introduced by the 2^ν-ary modulation signal set, which carries ν bits per signal. Therefore, an interleaver of length equal to a few times ν is enough to approach best performance [CTB2]. Note that this interleaver has nothing to do with the interleaver that a turbo code or a block product code would use.

9.5 Turbo trellis-coded modulation (TTCM)

Conceptually, there are various approaches to the combination of turbo codes, or product codes with interleaving, and digital modulation: Pragmatic coded modulation [LGB], turbo TCM with *symbol interleaving* [RW1, RW2] and turbo TCM with *bit interleaving* [BDMP2].

9.5.1 *Pragmatic turbo TCM*

Motivated by the extraordinary performance of turbo coding schemes, in 1994 [LGB], another pragmatic coded modulation scheme was introduced. Its block diagram is shown in Figure 124. The main feature is, as in pragmatic TCM, the use of the turbo encoder and decoder operating as in binary transmission mode. This requires careful computation of the *bit metrics*, as in the case of BICM.

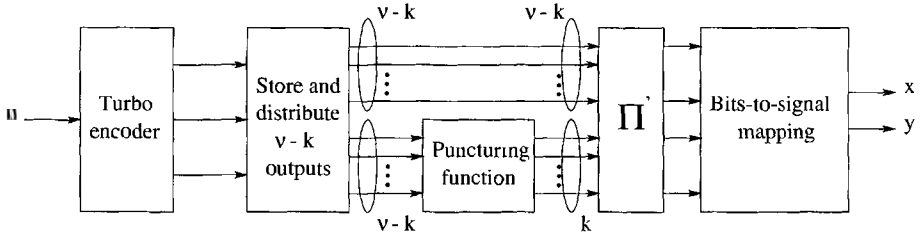


Figure 124 Combination of a turbo encoder and digital modulation [LGB]

9.5.2 *Turbo TCM with symbol interleaving*

In 1995, Robertson and Wörz proposed the use of recursive systematic convolutional encoders, such as those proposed by Ungerboeck [Ung1], as components in an overall coding system similar to that of turbo codes. A block diagram of this scheme is shown in Figure 125. As can be seen from the diagram, interleaving operates on symbols of ν bits, instead of on bits for binary turbo codes. There is a need to puncture *redundant symbols*, due to the two paths of modulated signal points. A careful component code (no parallel transitions) and interleaver design (even positions to even positions, odd-to-odd; or even-odd and odd-even) are required. In terms of iterative decoding, note that the systematic component cannot be separated from the extrinsic one since they are transmitted together in one symbol. However, the LLR can be separated into an a-priori and a systematic-and-extrinsic part. Care must be taken so that the information is not used more than once in the component decoders. This is the reason why redundant symbol puncturing, in the form of a selector, is needed at the output of the encoder [RW2]. Figure 126 shows a block diagram of an iterative decoder for turbo TCM.

9.5.3 *Turbo TCM with bit interleaving*

In 1996, Benedetto et al. [BDMP2] proposed symbol puncturing rules such that the outputs of the encoder contain the information bits only once. Moreover, as opposed to symbol interleaving and puncturing of redundant symbols, multiple *bit interleavers* were proposed. A

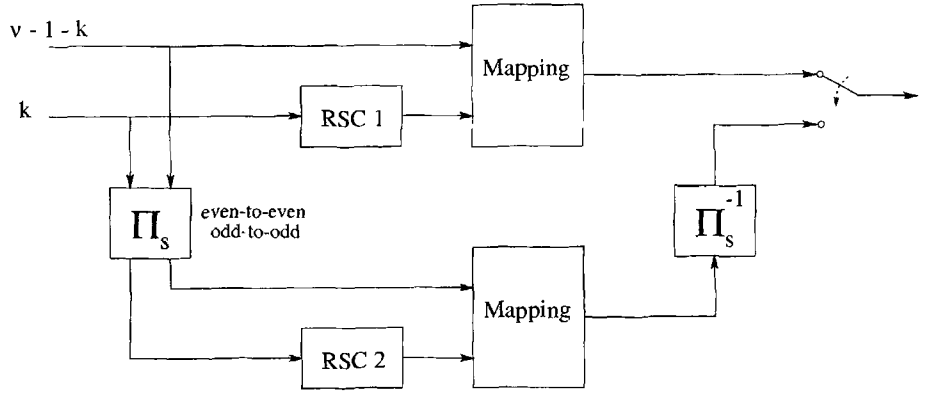


Figure 125 The encoder structure of a turbo TCM scheme with symbol interleaving.

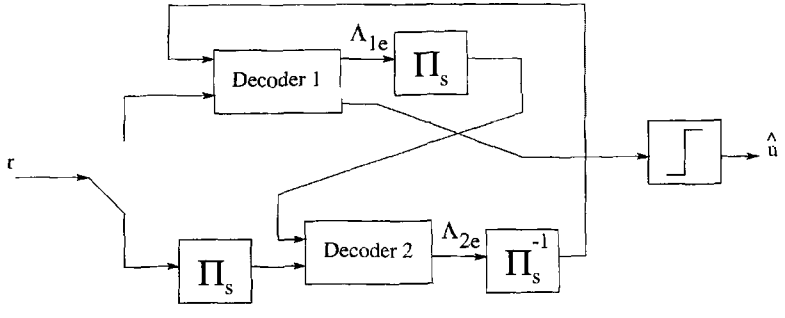


Figure 126 An iterative decoder for turbo TCM with symbol interleaving.

block diagram of the encoder structure is shown in Figure 127, for the case of two component codes.

MAP decoding and bit metrics

The decoder structure for turbo TCM with bit interleaving is similar to that of binary turbo codes. The main difference is that conversion of LLRs from bits to symbols, and from symbols to bits, needs to be performed between decoders [BDMP2, Vuc]. For decoding of turbo TCM with bit interleaving, the LLRs that are computed per bit need to be converted to a symbol level a-priori probability. Also, the a priori probabilities per symbol need to be converted to a bit level extrinsic LLR's.

This is done in the following way. Let $\mathcal{X}$ denote the 2^ν-ary modulation signal set. For a symbol $x(\bar{b}) \in \mathcal{X}$ with label $\bar{b} = (b_1, b_2, \cdots, b_\nu)$, the extrinsic information of bit b_i, $i = 1, 2, \cdots, \nu$, is computed as

$$\Lambda_e(b_i) = \log \left(\frac{\sum_{x(\bar{b}), b_i = 1} e^{\Lambda_e(x(\bar{b}))}}{\sum_{x(\bar{b}), b_i = 0} e^{\Lambda_e(x(\bar{b}))}} \right). \tag{9.16}$$

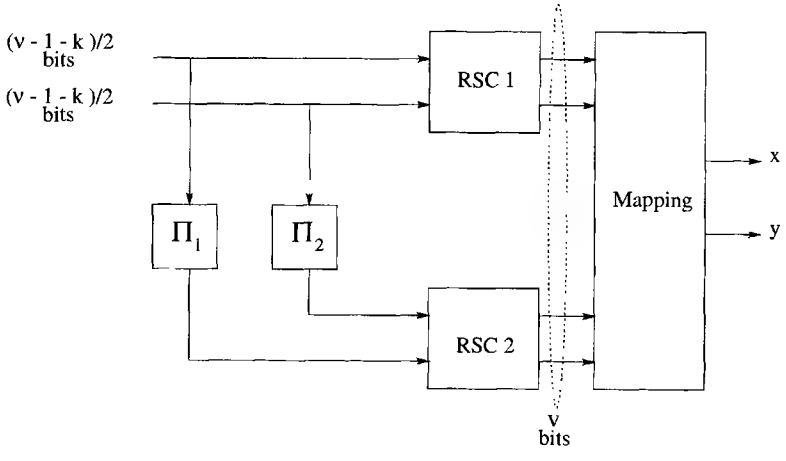


Figure 127 Encoder for turbo TCM with bit interleaving.

Similarly, the a-priori symbol probability can be computed from the extrinsic LLR at the bit level through the expression,

$$\Pr\left(\bar{b} = (b_1, b_2, \cdots, b_\nu)\right) = \prod_{i=1}^{\nu} \frac{e^{b_i \Lambda_e(b_i)}}{1 + e^{\Lambda_e(b_i)}}. \tag{9.17}$$

References

[BCJR] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Info. Theory*, vol. IT-20, pp. 284-287, Mar. 1974.

[BP] A. S. Barbulescu and S. S. Pietrobon, "Interleaver Design for Turbo Codes," *Elect. Letters*, vol. 30, no. 25, pp. 2107-2108, Dec. 1994.

[Bat] G. Battail, "A Conceptual Framework for Understanding Turbo Codes," *IEEE J. Sel. Areas in Comm.*, vol. 16, no. 2, pp. 245-254, Feb. 1998.

[BB] S. Benedetto and E. Biglieri, *Principles of Digital Transmission*, Kluwer Academic/Plenum Publishers, 1999.

[BM] S. Benedetto and G. Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schenes," *IEEE Trans. Info. Theory*, vol.42, no. 2, pp. 409-428, March 1996.

[BDMP1] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding," *IEEE Trans. Info. Theory*, vol. 44, no. 3, pp. 909-926, May 1998.

[BDMP2] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Parallel Concatenated Trellis Coded Modulation," *Proc. 1996 IEEE Int. Conf. Comm. (ICC'96)*, pp. 974-978, 1996.

[Ber1] E. R. Berlekamp, *Algebraic Coding Theory*, rev. ed., Aegean Park Press, 1984.

[Ber2] E. R. Berlekamp, "Bounded Distance + 1 Soft-Decision Reed-Solomon Decoding," *IEEE Trans. Info. Theory*, vol. 42, no. 3, pp. 704-720, May 1996.

[BG1] C. Berrou and A. Glavieux, "Reflections on the Prize Paper: 'Near Optimum Error-Correcting Coding and Decoding: Turbo Codes'," *IEEE Info. Theory Soc. News.*, vol. 48, no. 2, June 1998.

[BG2] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding," *IEEE Trans. Comm.*, vol. 44, no. 10, pp. 1261-1271, Oct. 1996.

[BGT] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," *Proc. 1993 IEEE Int. Conf. Comm. (ICC'93)*, pp. 1064-1070, Geneve, Switzerland, May 1993.

[BDMS] E. Biglieri, D. Divsalar, P. J. McLane and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, Macmillan Publishing, 1991.

[Blah] R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, 1984.

[BL] M. Blaum, "A Family of Efficient Burst-Correcting Array Codes," *IEEE Trans. Info. Theory*, vol. 36, no. 3, pp. 671-675, May 1990.

[BZ] E.L. Blokh and V.V. Zyablov, "Coding of Generalized Concatenated Codes," *Probl. Pered. Informatsii*, vol.10, no.1, pp. 45-50, 1974.

[BABSZ] M. Boo, F. Arguello, J. D. Bruguera, R. Doallo and E. L. Zapata, "High-Performance VLSI Architecture for the Viterbi Algorihtm," *IEEE Trans. Comm.*, vol. 45, no. 2, pp. 168-176, Feb. 1997.

[Bos] M. Bossert, *Channel Coding for Telecommunications*, John Wiley & Sons, 1999.

[BH] M. Breiling and J. B. Huber, "Combinatorial Analysis of the Minimum Distance of Turbo Codes," *IEEE Trans. Info. Theory*, vol. 47, no. 7, pp. 2737-2750, Nov. 2001.

[Bro] A. E. Brouwer and T. Verhoeff, "An Updated Table of Minimum-Distance Bounds for Binary Linear Codes," *IEEE Trans. Info. Theory*, vol. 39, no. 2, pp. 662-677, Mar. 1993.

[BW] H. O. Burton and E. J. Weldon, Jr., "Cyclic Product Codes," *IEEE Trans. Info. Theory*, vol. IT-11, no. 3, pp. 433-439, July 1965.

[CCG] J. B. Cain, G. C Clark and J. M. Geist, "Punctured Convolutional Codes of Rate $(n-1)/n$ and Simplified Maximum Likelihood Decoding," *IEEE Trans. Info. Theory*, vol. IT-25, pp. 97-100, Jan.1979.

[CTB1] G. Caire, G. Taricco and E. Biglieri, "Capacity of Bit-Interleaved Channels," *Elect. Letters*, vol. 32, pp. 1060-1061, June 1996.

[CTB2] G. Caire, G. Taricco and E. Biglieri, "Bit-Interleaver Coded Modulation," *IEEE Trans. Info. Theory*, vol. 44, no. 3, pp. 927-946, May 1998.

[CRKO] F. Carden, M.D. Ross, B.T. Kopp and W.P. Osborn, "Fast TCM Decoding: Phase Quantization and Integer Weighting," *IEEE Trans. Comm.*, vol. 42, no. 4, pp. 808-812, Apr. 1994.

[CY] C.-C. Chao and Y.-L. Yao, "Hidden Markov Models for the Burst Error Statistics of Viterbi Decoding," *IEEE Trans. Comm.*, vol. 44, no. 12, pp. 1620-1622, Dec. 1996.

[Cha] D. Chase, "A Class of Algorithms for Decoding Block Codes with Channel Measurement Information," *IEEE Trans. Info. Theory*, vol. IT-18, pp. 170-182, 1972.

[CK] P. Chaudhari and A. K. Khandani, "Using the Fourier Transform to Compute the Weight Distribution of a Binary Linear Block Code," *IEEE Comm. Let.*, vol. 5, no. 1, pp. 22-24, Jan. 2001.

[CFRU] S.-Y. Chung, G. D. Forney, Jr., T. J. Richardson and R. Urbanke, "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit," *IEEE Comm. Let.*, vol. 5, no. 2, pp. 58-60, Feb. 2001.

[Cla] G. Clark and J. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, 1981.

[Col] O. M. Collins, "The Subtleties and Intricacies of Building a Constraint Length 15 Convolutional Decoder," *IEEE Trans. Comm.*, vol. 40, no. 12, pp. 1810-1819, Nov. 1992.

[CoT] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, 1991.

[DFK1] Y. Desaki, T. Fujiwara and T. Kasami, "A Method for Computing the Weight Distribution of a Block Code by Using Its Trellis Diagram," *IEICE Tran. Fundamentals*, vol. E77-A, pp. 1230-1237, Aug. 1994.

[DFK2] Y. Desaki, T. Fujiwara and T. Kasami, "The Weight Distribution of Extended Binary Primitive BCH Codes of Length 128," *IEEE Trans. Info. Theory*, vol. 43, no. 4, pp. 1364-1371, July 1997.

[DYS] U. Dettmar, G. Yan and U. K. Sorger, "Modified Generalized Concatenated Codes and Their Application to the Construction and Decoding of LUEP Codes," *IEEE Trans. Info. Theory*, vol.41, no. 5, pp. 1499-1503, Sept. 1995.

[Dho] A. Dholakia, *Introduction to Convolutional Codes With Applications*, Kluwer, 1994.

[DDP] D. Divsalar, S. Dolinar and F. Pollara, "Iterative Turbo Decoder Analysis Based on Density Evolution," *IEEE J. Sel. Areas in Comm.*, vol. 19, no. 5, pp. 891-907, May 2001.

[DJM] D. Divsalar, H. Jin and R. J. McEliece, "Coding Theorems for Turbo-Like Codes," *Proc. 1998 Allerton Conf. on Commun., Control, and Computing*, pp. 210-219, Univ. Illinois; Urbana-Champaign, 1998.

[Div] D. Divsalar and F. Pollara, "Turbo Codes for Deep-Space Communications," *JPL TDA*

Progress Report 42-120, pp. 29-39, Feb. 1995.

[DP] D. Divsalar and F. Pollara, "Turbo Codes for PCS Applications," *Proc. 1993 IEEE Int. Conf. Comm. (ICC'93)*, pp. 1064-1070, Geneva, Switzerland, May 1993.

[DVB] EN 310 210, ETSI, "Digital Video Broadcasting (DVB): Framing Structure, Channel Coding and Modulation for Digital Satellite News Gathering (DSNG) and Other Contribution Applications by Satellite," Mar. 1997.

[EH] H. El Gamal and A. R. Hammons, Jr., "Analyzing the Turbo Decoder Using the Gaussian Approximation," *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp. 671-686, Feb. 2001.

[Eli1] P. Elias, "Error-Free Coding," *IRE Trans.*, vol. PGIT-4, pp. 29-37, 1954.

[Eli2] P. Elias, "Coding for Noisy Channels," *IRE Conv. Rec.*, vol. 3, pt. 4, pp. 37-46, 1955. Also in E.R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, pp. 48-55, IEEE Press, 1974.

[Eli3] P. Elias, "Error-Correcting Codes for List Decoding," *IEEE Trans. Info. Theory*, vol. 37, no. 1, pp. 5-12, Jan. 1991.

[FV] W. Feng and B. Vucetic, "A List Bidirectional Soft Output Decoder of Turbo Codes," *Proc. Int. Symp. Turbo Codes and Rel. Top.*, pp. 288-292, Brest, France, Sept. 3-5, 1997.

[For1] G. D. Forney, Jr., *Concatenated Codes*, MIT Press Reseach Monograph 37, 1966.

[For2] G,D, Forney, Jr., "On Decoding BCH Codes," *IEEE Trans. Info. Theory*, vol. IT-11, pp. 393-403, Oct. 1965. Also in E.R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, pp. 149-155, IEEE Press, 1974.

[For3] G. D. Forney, "Generalized Minimum Distance Decoding," *IEEE Trans. Info. Theory*, vol. IT-12, pp. 125-131, April 1966.

[For4] G. D. Forney, "Convolutional Codes I: Algebraic Structure," *IEEE Trans. Info. Theory*, vol. IT-16, no. 6, pp. 720-738, Nov. 1970.

[For5] G. D. Forney, Jr., "Burst Correcting Codes for the Classic Bursty Channel," *IEEE Trans. Comm. Tech.*, vol. COM-19, pp. 772-281, 1971.

[For6] G. D. Forney, Jr., "Coset Codes II: Binary Lattices and Related Codes," *IEEE Trans. Info. Theory*, vol. 24, no. 5, pp. 1152-1187, Sept. 1988.

[For7] G. D. Forney, "Codes on Graphs: Normal Realizations," *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp. 520-548, Feb. 2001.

[For8] G. D. Forney and G. Ungerboeck, "Modulation and Coding for Linear Gaussian Channels," *IEEE Trans. Info. Theory, Special Commemorative Issue*, vol. 44, no. 6, pp. 2384-2415, Oct. 1998.

[FBLH] M. P. C. Fossorier, F. Burkert, S. Lin and J. Hagenauer, "On the Equivalence Between SOVA and Max-Log-MAP Decodings," *IEEE Comm. Letters*, vol. 2, no. 5, pp. 137-139, May 1998.

[FLC] M. P. C. Fossorier, S. Lin and D. J. Costello, Jr., "On the Weight Distribution of Terminated Convolutional Codes," *IEEE Trans. Info. Theory*, vol. 45, no. 5, pp. 1646-1648, July 1999.

[FL1] M. P. C. Fossorier and S. Lin, "Soft-Decision Decoding on Linear Block Codes Based on Ordered Statistics," *IEEE Trans. Info. Theory*, vol. 41, no. 5, pp. 1379-1396, Sept. 1995.

[FL2] M. P. C. Fossorier and S. Lin, "Differential Trellis Decoding of Convolutional Codes," *IEEE Trans. Info. Theory*, vol. 46, no. 3, pp. 1046–1053, May 2000.

[FL3] M. P. C. Fossorier and S. Lin, "Some Decomposable Codes: The $|a+x|b+x|a+b+x|$ Construction," *IEEE Trans. Info. Theory*, vol. 43, no. 5, pp. 1663-1667, Sept. 1997.

[FL4] M. P. C. Fossorier and S. Lin, "Complementary Reliability-Based Soft-Decision Decoding of Linear Block Codes Based on Ordered Statistics," *IEEE Trans. Info. Theory*, vol. 42, no. 5, pp. 1667-1672, Sep. 1997.

[FL5] M. P. C. Fossorier and S. Lin, "Soft-Input Soft-Output Decoding of Linear Block Codes

Based on Ordered Statistics," *Proc. 1998 IEEE Global Telecomm. Conf. (GLOBECOM'98)*, pp. 2828-2833, Sydney, Australia, Nov. 1998.

[FLR] M. P. C. Fossorier, S. Lin and D. Rhee, "Bit-Error Probability for Maximum-Likelihood Decoding of Linear Block Codes and Related Soft-Decision Decoding Methods," *IEEE Trans. Info. Theory*, vol. 44, no. 7, pp. 3083-3090, Nov. 1998.

[FMH] M. P. C. Fossorier, M. Mihaljević and H. Imai, "Reduced Complexity Iterative Decoding of Low-Density Parity Check Codes Based on Belief Propagation," *IEEE Trans. Comm.*, vol. 47, no. 5, pp. 673-680, May 1999.

[FK] B. J. Frey and F. R. Kschischang, "Early Detection and Trellis Splicing: Reduced-Complexity Iterative Decoding," *IEEE J. Sel. Areas in Comm.*, vol. 16, no. 2, pp. 153-159, Feb. 1998.

[FKKL] T. Fujiwara, T. Kasami, A. Kitai and S. Lin, "On the Undetected Error Probability of Shortened Hamming Codes," *IEEE Trans. Comm.*, vol. COM-33, pp. 570-574, June 1985.

[Gal] R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Info. Theory*, vol. 8, no. 1, pp. 21-28, Jan. 1962.

[GPB] R. Garello, P. Perleoni and S. Benedetto, "Computing the Free Distance of Turbo Codes and Serially Concatenated Codes with Interleavers: Algorithms and Applications," *IEEE J. Sel. Areas in Comm.*, vol. 19, no. 5, pp. 800-812, May 2001.

[GG] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, 1999.

[Gol] M. J. E. Golay, "Notes on Digital Coding" *Proc. IRE.*, vol. 37, p. 657, June 1949. Also in *Key Papers in the Development of Coding Theory*, E. R. Berlekamp, ed., p. 13, IEEE Press, 1974.

[Glc] J. D. Golić, "Iterative Optimum Symbol-by-Symbol Decoding and Fast Correlation Attacks," *IEEE Trans. Info. Theory*, vol. 47, no. 7, pp. 3040-3049, Nov. 2001.

[GZ] D. Gorenstein and N. Zierler, "A Class of Error Correcting Codes in p^m Symbols," *J. SIAM*, vol. 9, pp. 207-214, June 1961.

[Gur] V. Guruswami and M. Sudan, "Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes," *IEEE Trans. Info. Theory*, vol. 45, no. 6, pp. 1757-1767, Sep. 1999.

[Hag] J. Hagenauer, "Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications," *IEEE Trans. Comm.*, vol. 36, no. 4, pp. 389-400, April 1988.

[HH] J. Hagenauer and P. Hoher, "A Viterbi Algorithm with Soft-Decision Outputs and Its Applications," *Proc. 1989 IEEE Global Telecomm. Conf. (GLOBECOM'89)*, pp. 47.1.1-47.1.7, Dallas, Texas, 1989.

[HOP] J. Hagenauer, E. Offer and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Info. Theory*, vol. 42, no. 2, pp. 429-445, Mar. 1996.

[HaW] E. K. Hall and S. G. Wilson, "Stream-Oriented Turbo Codes," *IEEE Trans. Info. Theory*, vol. 47, no. 5, pp. 1813-1831, July 2001.

[Ham] R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.*, vol. 29, pp. 147-160, 1950. Also in *Key Papers in the Development of Coding Theory*, E. R. Berlekamp, ed., pp. 9-12, IEEE Press, 1974.

[Hay] S. Haykin, *Digital Communications*, John Wiley and Sons, 1988.

[HeW] C. Heegard and S. B. Wicker, *Turbo Coding*, Kluwer Academic Press, 1999.

[HEK] A. P. Hekstra, "An Alternative to Metric Rescaling in Viterbi Decoder," *IEEE Trans. Comm.*, vol. 37, no. 11, pp. 1220-1222, Nov. 1989.

[Her] I. N. Herstein, *Topics in Algebra*, 2nd ed., John Wiley and Sons, 1975.

[HEM] J. Hokfelt, O. Edfors and T. Maseng, "A Turbo Code Interleaver Design Criterion Based on the Performance of Iterative Decoding," *IEEE Comm. Let.*, vol. 5, no. 2, pp. 52-54, Feb. 2001.

[HM] B. Honay and G.S. Markarian, *Trellis Decoding of Block Codes: A Practical Approach*, Kluwer, 1996.

[HMF] B. Honary, G.S. Markarian and P. G. Farell, "Generalised Array Codes and Their Trellis Structure," *Elect. Letters*, vol. 28, no. 6, pp. 541-542, Mar. 1993.

[Hsia] M.Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes," *IBM J. Res. and Dev.*, vol. 14, July 1970.

[IH] H. Imai and S. Hirakawa, "A New Multilevel Coding Method Using Error-Correcting Codes," *IEEE Trans. Info. Theory*, vol. IT-23, no. 3, pp. 371-377, May 1977.

[IFMLI] M. Isaka, M. P. C. Fossorier, R. H. Morelos-Zaragoza, S. Lin and H. Imai, "Multilevel Coded Modulation for Unequal Error Protection and Multistage Decoding. Part II: Asymmetric Constellations," *IEEE Trans. Comm.*, vol. 48, no. 5, pp. 774-784, May 2000.

[ISTC1] *Proceedings of the International Symposium on Turbo Codes and Related Topics*, Brest, France, September 3-5, 1997.

[ISTC2] *Proceedings of the Second International Symposium on Turbo Codes and Related Topics*, Brest, France, September 4-7, 2000.

[ITCG] Special Issue on Codes and Graphs and Iterative Decoding Algorithms, *IEEE Trans. Info. Theory*, vol. 47, no. 2, Feb. 2001.

[Jer] M. C. Jeruchim, P. Balaban and K. S. Shanmugan, *Simulation of Communication Systems*, Plenum Press, 1992.

[Joh] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*, IEEE Press, 1999.

[JSAC1] Special Issue on the turbo principle: from theory to practice. Part I, *IEEE J. Sel. Areas in Comm.*, vol. 19, no. 5, May 2001.

[JSAC2] Special Issue on the turbo principle: from theory to practice. Part II, *IEEE J. Sel. Areas in Comm.*, vol. 19, no. 9, Sept. 2001.

[Kam] N. Kamiya, "On Algebraic Soft-Decision Decoding Algorithms for BCH Codes," *IEEE Trans. Info. Theory*, vol. 47, no. 1, pp. 45-58, Jan. 2001.

[KNIH] T. Kaneko, T. Nishijima, H. Inazumi and S. Hirasawa, "An Efficient Maximum-Likelihood Decoding Algorithm for Linear Block Codes with Algebraic Decoder," *IEEE Trans. Info. Theory*, vol. 40, no. 2, pp. 320-327, Mar. 1994.

[Kas1] M. Kasahara, Y. Sugiyama, S. Hirasawa and T. Namekawa, "A New Class of Binary Codes Constructed on the Basis of BCH Codes," *IEEE Trans. Info. Theory*, vol. IT-21, pp. 582-585, 1975.

[Kas2] M. Kasahara, Y. Sugiyama, S. Hirasawa and T. Namekawa, "New Classes of Binary Codes Constructed on the Basis of Concatenated and Product Codes," *IEEE Trans. Info. Theory*, vol. IT-22, no. 4, pp. 462-468, July 1976.

[Kasm] T. Kasami, "A Decoding Procedure for Multiple-Error-Correcting Cyclic Codes," *IEEE Trans. Info. Theory*, vol. IT-10, pp. 134-138, 1964.

[KLP] T. Kasami, S. Lin and W. W. Peterson, "Polynomial Codes," *IEEE Trans. Info. Theory*, vol. IT-14, no. 6, pp. 807-814, Nov. 1968.

[KKTFL] T. Kasami, T. Koumoto, T. Takata, T. Fujiwara and S. Lin, "The Least Stringent Sufficient Condition on the Optimality of Suboptimally Decoded Codewords," *Proc. 1995 IEEE Int. Symp. Info. Theory (ISIT'95)*, p. 470, Whistler, Canada, 1995.

[Kaz] P. Kazakov, "Fast Calculation of the Number of Minimum-Weight Words of CRC Codes," *IEEE Trans. Info. Theory*, vol. 47, no. 3, pp. 1190-1195, March 2001.

[KLF] Y. Kuo, S. Lin and M. P. C. Fossorier, "Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results," *IEEE Trans. Info. Theory*, vol. 47, no. 7, pp. 2711-2736, Nov. 2001.

[KFL] R. R. Kschischang, B. J. Frey and H.-A. Loeliger, "Factor Graphs and the Sum-Product

Algorithm," *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp. 498-519, Feb. 2001.

[KV] R. Koetter and A. Vardy, "Algebraic Soft-Decision Decoding of Reed-Somon Codes," *Proc. 2000 IEEE Int. Symp. Info. Theory (ISIT'00)*, p. 61, Sorrento, Italy, June 25-30, 2000.

[Lee] L. H. C. Lee, *Convolutional Coding: Fundamentals and Applications*, Artech House, 1997.

[LGB] S. Le Goff, A. Glavieux and C. Berrou, "Turbo-Codes and High-Spectral Efficiency Modulation," *Proc. 1994 IEEE Int. Conf. Comm. (ICC'94)*, pp. 645-649, 1994.

[LC] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, 1983.

[LKFF] S. Lin, T. Kasami, T. Fujiwara and M. Fossorier, *Trellises and Trellis-Based Decoding Algorithms for Linear Block Codes*, Kluwer Academic Press, 1998.

[LYHH] J. Lodge, R. Young, P. Hoeher and J. Hagenauer, "Separable MAP 'Filters' for the Decoding of Product and Concatenated Codes," *Proc. 1993 IEEE Int. Conf. Comm. (ICC'93)*, pp. 1740-1745, May 1993.

[Mac] D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices," *IEEE Trans. Info. Theory*, vol. 45, no. 2, pp. 399-432, Mar. 1999.

[MN] D. J. C. MacKay and R. M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electronics Letters*, vol. 32, pp. 1645-1646, 1996.

[MS] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, 1977.

[Man] D. Mandelbaum, "Decoding Beyond the Designed Distance of Certain Algebraic Codes," *Info. and Control*, vol. 35, pp. 209-228, 1977.

[MT] P. A. Martin and D. P. Taylor, "On Adaptive Reduced-Complexity Iterative Decoding," *Proc. 2000 IEEE Global Telecomm. Conf. (GLOBECOM'00)*, pp. 772 -776, 2000.

[Mas1] J. L. Massey, *Threshold Decoding*, MIT Press, 1963.

[Mas2] J.L. Massey, "Shift Register Synthesis and BCH Decoding," *IEEE Trans. Info. Theory*, vol. IT-15, no. 1, pp. 122-127, Jan. 1969.

[Mas3] J. L. Massey, "Coding and Modulation in Digital Communications," *Proc. Int. Zurich Seminar on Dig. Comm.*, pp. E2(1)-E2(4), Zurich, Switzerland, 1974.

[Mas4] J. L. Massey, "The How and Why of Channel Coding," *Proc. Int. Zurich Seminar on Dig. Comm.*, pp. 67-73, Zurich, Switzerland, 1984.

[MW] B. Masnick and J. Wolf, "On Linear Unequal Error Protection Codes," *IEEE Trans. Info. Theory*, vol. IT-13, no. 4, pp. 600-607, July 1967.

[McE] R. J. McEliece, *The Theory of Information and Coding*, Addison-Wesley, 1977.

[MMC] R. J. McEliece, D. J. C. MacKay and J.-F. Cheng, "Turbo Decoding as an Instance of Pearls' 'Belief Propagation' Algorithm," *IEEE J. Sel. Areas in Comm.*, vol. 16, no. 2, pp. 140-152, Feb. 1998.

[Meg] J.E. Meggit, "Error Correcting Codes for Correcting Bursts of Errors," *IBM J. Research Develop.*, no. 4, pp. 329-334, 1960.

[ML] A.M. Michelson and A.H. Levesque, *Error-Control Techniques for Digital Communications*, John Wiley and Sons, 1985.

[MG] M. J. Mihaljević and J. D. Golić, "A Comparison of Cryptoanalitic Principles Based on Iterative Error Correction," *Advances in Cryptology — EUROCRYPT'91 (Lecture Note in Computer Science)*, vol. 547, pp. 527-531, Springer-Verlag, 1991.

[MFKL] R. H. Morelos-Zaragoza, T. Fujiwara, T. Kasami and S. Lin, "Constructions of Generalized Concatenated Codes and Their Trellis-Based Decoding Complexity," *IEEE Trans. Info. Theory*, vol. 45, no. 2, pp. 725-731, Mar. 1999.

[MI] R. H. Morelos-Zaragoza and H. Imai, "Binary Multilevel Convolutional Codes with Unequal Error Protection Capabilities," *IEEE Trans. Comm.*, vol. 46, no. 7, pp. 850-853, July

1998.

[MM] R. H. Morelos-Zaragoza and A. Mogre, "A Two-Stage Decoder for Pragmatic Trellis-Coded M-PSK Modulation Using a Symbol Transformation," *IEEE Trans. Comm.*, vol. 49, no. 9, pp. 1501-1505, Sept. 2001.

[MFLI] R. H. Morelos-Zaragoza, M. P. C. Fossorier, S. Lin and H. Imai, "Multilevel Coded Modulation for Unequal Error Protection and Multistage Decoding. Part I: Symmetric Constellations," *IEEE Trans. Comm.*, vol. 48, no. 2, pp. 204-213, Feb. 2000.

[Mor] J. M. Morris, "Burst Error Statistics of Simulated Viterbi Decoded BPSK on Fading and Scintillating Channels," *IEEE Trans. Comm.*, vol. 40, no. 1, pp. 34-41, Jan. 1992.

[OCC] I. M. Onyszchuk, K.-M. Cheung and O. Collins, "Quantization Loss in Convolutional Decoding," *IEEE Trans. Comm.*, vol. 41, no. 2, pp. 261-265, Feb. 1993.

[Prl] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.

[PSC] L. C. Perez, J. Seghers and D. J. Costello, Jr., "A Distance Spectrum Interpretation of Turbo Codes," *IEEE Trans. Info. Theory*, vol. 42, no. 6, pp. 1698-1709, Nov. 1996.

[PW] W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes*, 2nd ed., MIT Press, 1972.

[Pet] W.W. Peterson, "Encoding and Error-Correction Procedures for the Bode-Chaudhuri Codes," *IRE Trans. Info. Theory*, vol. IT-6, pp. 459-470, Sept. 1960. Also in E.R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, pp. 109-120, IEEE Press, 1974.

[PP] A. Picart and R. M. Pyndiah, "Adapted Iterative Decoding of Product Codes," *Proc. 1995 IEEE Global Telecomm. Conf. (GLOBECOM'95)*, pp. 2357-2362, 1995.

[Plo] M. Plotkin, "Binary Codes with Specified Minimum Distances," *IEEE Trans. Info. Theory*, vol. IT-6, pp. 445-450, 1960.

[Pre] O. Pretzel, *Codes and Algebraic Curves*, Oxford Lecture Series in Mathematics and Its Applications, 8, 1998.

[Pro] J. G. Proakis, *Digital Communications*, 3rd ed., McGraw-Hill, 1995.

[PS] M.B. Pursley and J.M. Shea, "Bit-by-Bit Soft-Decision Decoding of Trellis-Coded M-DPSK Modulation," *IEEE Comm. Letters*, vol. 1, no. 5, pp. 133-135, Sept. 1997.

[Pyn] R. M. Pyndiah, "Near-Optimum Decoding of Product Codes: Block Turbo Codes," *IEEE Trans. Comm.*, vol. 46, no. 8, pp. 1003-1010, Aug. 1998.

[PGPJ] R. M. Pyndiah, A. Glavieux, A. Picart and S. Jacq, "Near Optimum Decoding of Product Codes," *Proc. 1994 IEEE Global Telecomm. Conf. (GLOBECOM'94)*, vol. 1, pp. 339-343, San Francisco, CA, Dec. 1994.

[Ram] J. L. Ramsey, "Realization of Optimum Interleavers," *IEEE Trans. Info. Theory*, vol. IT-16, no. 3, pp. 338-345, May 1970.

[RR] S. M. Reddy and J. P. Robinson, "Random Error and Burst Correction by Iterated Codes," *IEEE Trans. Info. Theory*, vol. IT-18, no. 1, pp. 182-185, Jan. 1972.

[RC] I. S. Reed and X. Chen, *Error-Control Coding for Data Networks*, Kluwer Academic Press, 1999.

[RS] I. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *SIAM J. Appl. Math.*, Vol. 8, pp. 300-304, 1960. Also in E.R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, pp. 70-71, IEEE Press, 1974.

[RSU] T. J. Richardson, M. A. Shokrollahi and R. L. Urbanke, "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes," *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp. 619-637, Fe. 2001.

[RU] T. J. Richardson and R. L. Urbanke, "The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding," *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp. 599-618, Feb. 2001.

[Rob] P. Robertson, "Improving Decoder and Code Structure of Parallel Concatenated Recursive

Systematic (Turbo) Codes," *Proc. IEEE Int. Conf. Univ. Per. Comm. (ICUPC'94)*, pp. 183-187, 1994.

[RVH] P. Robertson, E. Villebrun and P. Hoeher, "A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain," *Proc. 1995 IEEE Int. Conf. Comm. (ICC'95)*, pp. 1009-1013, 1995.

[RW1] P. Robertson and T. Wörz, "Coded Modulation Scheme Employing Turbo Codes," *Electronics Letters*, vol. 31, pp. 1546-1547, Aug. 1995.

[RW2] P. Robertson and T. Wörz, "Bandwidth-Efficient Turbo Trellis-Coded Modulation Using Punctured Component Codes," *IEEE J. Sel. Areas in Comm.*, vol. 16, no. 2, pp. 206-218, Feb. 1998.

[SSSN] H. R. Sdajadpour, N. J. A. Sloane, M. Salehi and G. Nebe, "Interleaver Design for Turbo Codes," *IEEE J. Sel. Areas in Comm.*, vol. 19, no. 5, pp. 831-837, May 2001.

[SB] G. Schnabl and M. Bossert, "Soft-Decision Decoding of Reed-Muller Codes as Generalized Multiple Concatenated Codes," *IEEE Trans. Info. Theory*, vol. 41, no. 1, pp. 304-308, Jan. 1995.

[Schr] P. Schramm, "Multilevel Coding with Independent Decoding on Levels for Efficient Communication on Static and Interleaved Fading Channels," *Proc. IEEE 1997 Int. Symp. Per., Ind. and Mob. Radio Comm. (PIMRC'97)*, pp. 1196-1200, 1997.

[Sha] C. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379-423 and 623-656, 1948. Also in *Key Papers in the Development of Information Theory*, D. Slepian, ed., pp. 5-29, IEEE Press, 1974.

[SLF] R. Shao, S. Lin and M. P. C. Fossorier, "Two Simple Stopping Criteria for Turbo Decoding," *IEEE Trans. Comm.*, vol. 47, no. 8, pp. 1117-1120, Aug. 1999.

[Sin] R.C. Singleton, "Maximum Distance q-nary Codes," *IEEE Trans. Info. Theory*, vol. IT-10, pp. 116-118, 1964.

[SS] M. Sipser and D. A. Spielman, "Expander Codes," *IEEE Trans. Info. Theory*, vol. 42, no. 6, pp. 1710-1722, Nov. 1996.

[Sle] D. Slepian, "A Class of Binary Signaling Alphabets," *Bell. Sys. Tech. J.*, vol. 35, pp. 203-234, Jan. 1956.

[Slo] N. J. A. Sloane, S. M. Reddy and C.-L. Chen, "New Binary Codes," *IEEE Trans. Info. Theory*, vol. IT-18, no. 4, pp. 503-510, July 1972.

[Sud] M. Sudan, "Decoding of Reed-Solomon Codes Beyond the Error-Correction Bound," *J. Complexity*, vol. 12, pp. 180-193, Dec. 1997.

[SKHN] Y. Sugiyama, Y. Kasahara, S. Hirasawa and T. Namekawa, "A Method for Solving Key Equation for Goppa Codes," *Info. and Control*, vol. 27, pp. 87-99, 1975.

[Sug] Y. Sugiyama, M. Kasahara, S. Hirasawa and T. Namekawa, "Further Results on Goppa Codes and Their Applications to Constructing Efficient Binary Codes," *IEEE Trans. Info. Theory*, vol. IT-22, pp. 518-526, 1978.

[TP] D. J. Taipale and M. B. Pursley, "An Improvement to Generalized Minimum Distance Decoding," *IEEE Trans. Info. Theory*, vol. 37, no. 1, pp. 167-172, Jan. 1991.

[TYFKL] T. Takata, Y. Yamashita, T. Fujiwara, T. Kasami and S. Lin, "On a suboptimum decoding of decomposable block codes," *IEEE Trans. Info. Theory*, vol. 40, no. 5, pp. 1392-1406, Sept. 1994.

[TC] O. Y. Takeshita and D. J. Costello, Jr., "New Deterministic Interleaver Designs for Turbo Codes," *IEEE Trans. Info. Theory*, vol. 46, no. 6, pp. 1988-2006, Sept. 2000.

[TLFL] H. Tang, Y. Liu, M. P. C. Fossorier and S. Lin, "On Combining Chase-2 and GMD Decoding Algorithms for Nonbinary Block Codes," *IEEE Comm. Letters*, vol. 5, no. 5, pp. 209-211, May 2001.

[Tan] R. M. Tanner, "A Recursive Approach to Low Complexity Codes," *IEEE Trans. Info.*

Theory, vol. IT-27, no. 5, pp. 533-547, Sept. 1981.

[tBr1] S. ten Brink , "Convergence of iterative decoding," *Electronics Letters*, vol. 35, pp. 806-808, May 1999.

[tBr2] S. ten Brink , "Convergence Behaviour of Iteratively Decoded Parallel Concatenated Codes," submitted to *IEEE Trans. Comm.*, Mar. 2000.

[TKK] H. Tokushige, T. Koumoto and T. Kasami, "An Improvement to GMD-like Decoding Algorithms," *Proc. 2000 IEEE Int. Symp. Info. Theory (ISIT'00)*. p. 396, Sorrento, Italy, 2000.

[Ung1] G. Ungerboeck, "Channel Coding with Multilevel/Phase Signals," *IEEE Trans. Info. Theory*, vol. IT-28, no. 1, pp. 55-67, Jan. 1982.

[Ung2] G. Ungerboeck, "Trellis-Coded Modulation with Redundant Signal Sets I: Introduction," *IEEE Comm. Mag.*, vol. 25, no. 2, pp. 5-11, Feb. 1987.

[Ung3] G. Ungerboeck, "Trellis-Coded Modulation with Redundant Signal Sets II: State of the Art," *IEEE Comm. Mag.*, vol. 25, no. 2, pp. 12-21, Feb. 1987.

[Van] W.J. van Gils, "Two Topics on Linear Unequal Error Protection Codes: Bounds on Their Length and Cyclic Code Classes," *IEEE Trans. Info. Theory*, vol. IT-29, no. 6, pp. 866-876, Nov. 1983.

[VO] S. A. Vanstone and P. C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic Publishers, 1989.

[Vit1] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Info. Theory*, vol. IT-13, pp. 260-269, April 1967.

[Vit2] A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," *IEEE Trans. Comm.*, vol. COM-19, no. 4, pp. 751-772, 1971.

[Vit3] A. J. Viterbi, "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes," *IEEE J. Sel. Areas in Comm.*, vol. 16, no. 2, pp. 260-264, Feb. 1998.

[Vit4] A.J. Viterbi, J.K. Wolf, E. Zehavi and R. Padovani, "A Pragmatic Approach to Trellis-Coded Modulation," *IEEE Comm. Mag.*, pp. 11-19, July 1989.

[ViOm] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, 1979.

[Vuc] B. Vucetic and J. Yuan, *Turbo Codes: Principles and Applications*, Kluwer Academic, 2000.

[WFH] U. Wachsmann, R. F. H. Fischer and J. B. Huber, "Multilevel Codes: Theoretical Concepts and Practical Design Rules," *IEEE Trans. Info. Theory*, vol. 45, no. 5, pp. 1361-1391, July 1999.

[WM] B. E. Wahlen and C. Y. Mai, "Turbo Coding Applied to Pragamtic Trellis-Coded Modulation," *IEEE Comm. Letters*, vol. 4, no. 2, pp. 65-67, Feb. 2000.

[Wel] E. J. Weldon, Jr., "Decoding Binary Block Codes on Q-ary Output Channels," *IEEE Trans. Info. Theory*, vol. IT-17, no. 6, pp. 713-718, Nov. 1971.

[Wib] N. Wiberg, "Codes and Decoding on General Graphs," Ph.D. dissertation, Dept. Elect. Eng., Linköping Univ., 1996.

[Wic] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice-Hall, 1995.

[WB] S.B. Wicker and V.K. Bhargava, *Reed-Solomon Codes and Their Applications*, IEEE Press, 1994.

[Wil] S. G. Wilson, *Digital Modulation and Coding*, Prentice-Hall, 1996.

[Wol] J. K. Wolf, "Efficient Maximum-Likelihood Decoding of Linear Block Codes Using a Trellis," *IEEE Trans. Info. Theory*, vol. IT-24, no. 1, pp. 76-80, Jan. 1978.

[WV] J.K. Wolf and A. J. Viterbi, "On the Weight Distribution of Linear Block Codes formed From Convolutional Codes," *IEEE Trans. Comm.*, vol. 44, no. 9, pp. 1049-1051, Sep. 1996.

[WJ] J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*, John Wiley and Sons, 1965.

[YI] K. Yamaguchi and H. Imai, "A New Block Coded Modulation Scheme and Its Soft Decision Decoding," *Proc. 1993 IEEE Inter. Symp. Info. Theory (ISIT'93)*, p. 64, 1993.

[YKH] Y. Yasuda, K. Kashiki and Y. Hirata, "High-Rate Punctured Convolutional Codes for Soft Decision Viterbi Decoding," *IEEE Trans. Comm.*, vol. COM-32, no. 3, pp. 325-328, March 1984.

[ZW] E. Zehavi and J.K. Wolf, "P^2 Codes: Pragmatic Trellis Codes Utilizing Punctured Convolutional Codes," *IEEE Communications Magazine*, pp. 94-99, Feb. 1995.

[ZP] V. V. Zyablov and M. S. Pinsker, "Estimation of the Error-Correction Complexity for Gallager Low-Density Codes," *Prob. Pered. Inform.*, vol. 11, no. 1, pp. 26-36, 1975.

[Zin] V. A. Zinov'ev, "Generalized Cascade Codes," *Probl. Pered. Informatsii*, vol. 12, no. 1, pp. 5-15, 1976.

Appendix A

Weight distributions of extended BCH codes

In this appendix, the weight distributions of all extended BCH codes of length up to 128 are presented. The first row of a table indicates the parameters of the code "n,k,d" (this is also the name of a file containing the weight distribution in the ECC web site.) Subsequent rows of a table list the weight w and the number of codewords of this weight A_w. These codes are symmetric, in the sense that the relation $A_w = A_{n-w}$, for $0 \leq w \leq n/2$, holds. Consequently, only half of the distribution is listed.

A.1 Length 8

```
wd.8.1.8
8 1

wd.8.4.4
4 14
8 1

wd.8.7.2
2 28
4 70
```

A.2 Length 16

```
wd.16.05.08
8  30

wd.16.07.06
6  48
8  30

wd.16.11.04
4  140
```

```
6  448
8  870

wd.16.15.02
2  120
4  1820
6  8008
8  12870
```

A.3 Length 32

```
wd.32.06.16
16 62

wd.32.11.12
12 496
16 1054

wd.32.16.08
8  620
12 13888
16 36518

wd.32.21.06
6  992
8  10540
10 60512
12 228160
14 446400
16 603942

wd.32.26.04
4  1240
6  27776
8  330460
10 2011776
12 7063784
14 14721280
16 18796230

wd.32.31.02
2  496
4  35960
6  906192
8  10518300
10 64512240
12 225792840
14 471435600
16 601080390
```

A.4 Length 64

```
wd.64.07.32
32 126

wd.64.10.28
28 448
32 126

wd.64.16.24
24 5040
28 12544
32 30366

wd.64.18.22
22 4224
24 5040
26 24192
28 12544
30 69888
32 30366

wd.64.24.16
16 2604
18 10752
22 216576
24 291648
26 1645056
28 888832
30 4419072
32 1828134

wd.64.30.14
14 8064
16 30828
18 631680
20 1128960
22 14022144
24 14629440
26 105057792
28 65046016
30 282933504
32 106764966

wd.64.36.12
12 30240
14 354816
16 3583020
18 27105792
20 145061280
22 603113472
```

```
24 1853011776
26 4517259264
28 8269968448
30 12166253568
32 13547993382

wd.64.39.10
10 13888
12 172704
14 2874816
16 29210412
18 214597824
20 1168181280
22 4794749760
24 14924626752
26 35889146496
28 66620912960
30 96671788416
32 109123263270

wd.64.45.08
8  27288
10 501760
12 12738432
14 182458368
16 1862977116
18 13739292672
20 74852604288
22 306460084224
24 956270217000
26 2294484111360
28 4268285380352
30 6180152832000
32 6991765639110

wd.64.51.06
6  20160
8  1067544
10 37051840
12 801494400
14 11684617344
16 119266575708
18 879321948288
20 4789977429888
22 19616032446528
24 61193769988008
26 146864398476096
28 273137809339136
30 395577405119232
32 447418802536902
```

```
wd.64.57.04
4  10416
6  1166592
8  69194232
10 2366570752
12 51316746768
14 747741998592
16 7633243745820
18 56276359749120
20 306558278858160
22 1255428754917120
24 3916392495228360
26 9399341113166592
28 17480786291963792
30 25316999607653376
32 28634752793916486

wd.64.63.02
2  2016
4  635376
6  74974368
8  4426165368
10 151473214816
12 3284214703056
14 47855699958816
16 488526937079580
18 3601688791018080
20 19619725782651116
22 80347448443237936
24 250649105469666110
26 601557853127198720
28 1118770292985240200
30 1620288010530347000
32 1832624140942591500
```

A.5 Length 128

```
wd.128.008.064
 64  254

wd.128.015.056
 56 8128
 64 16510

wd.128.022.048
 48 42672
 56 877824
 64 2353310

wd.128.029.044
```

```
 44 373888
 48 2546096
 52 16044672
 56 56408320
 60 116750592
 64 152623774

wd.128.036.032
 32 10668
 36 16256
 40 2048256
 44 35551872
 48 353494848
 52 2028114816
 56 7216135936
 60 14981968512
 64 19484794406

wd.128.043.032
 32 124460
 36 8810752
 40 263542272
 44 4521151232
 48 44899876672
 52 262118734080
 56 915924097536
 60 1931974003456
 64 2476672341286

wd.128.050.028
 28 186944
 32 19412204
 36 113839296
 40 33723852288
 44 579267441920
 48 5744521082944
 52 33558415333632
 56 117224663972352
 60 247312085243776
 64 31699236111910

wd.128.057.024
 24 597408
 28 24579072
 32 2437776684
 36 141621881856
 40 4315318568736
 44 74150180302848
 48 73528925007168
 52 4295496356229120
 56 15004724612905792
```

```
 60 31655991621445632
 64 4574965317267238

wd.128.064.022
 22 243840
 24 6855968
 26 107988608
 28 1479751168
 30 16581217536
 32 161471882796
 34 1292241296640
 36 9106516329984
 38 53383279307904
 40 278420690161824
 42 1218666847725184
 44 4782630191822848
 46 15858705600596992
 48 47425684161326912
 50 120442185147493376
 52 277061634654099456
 54 543244862505775360
 56 967799721857135168
 58 1473287478189735168
 60 2041819511308530688
 62 2421550630907043328
 64 2617075886216910118

wd.128.071.020
 20 2674112
 22 37486336
 24 839699616
 26 13825045248
 28 188001347136
 30 2140095182336
 32 20510697927468
 34 166689980438016
 36 1156658661471040
 38 6886497209935616
 40 35363776220195360
 42 157207798773129984
 44 607468163067994304
 46 2045773679068686336
 48 6023796954778012480
 50 15537040516548126720
 52 35191124114633006464
 54 70078589269156969984
 56 122925566952088660288
 58 190054082758956107264
 60 259342737902840355456
 62 312380032198035579904
 64 332409207867786543910
```

```
wd.128.078.016
 16 387096
 18 5462016
 20 213018624
 22 539859840
 24 107350803840
 26 1766071867392
 28 24074650400768
 30 273932927993856
 32 2625267567169884
 34 2133648518951040
 36 14805286631892608
 38 881470039149213696
 40 4526561735332554624
 42 20122606565844068352
 44 77755925658495682560
 46 261859003134276581376
 48 771046023044966543784
 50 1988741249124011372544
 52 4504463828911859699712
 54 8970059328813665832960
 56 15734472710169831412480
 58 2432692269013718774l696
 60 3319587221944924483584
 62 3998464407989233708646l
 64 42548378876302513514950

wd.128.085.014
 14 341376
 16 22121368
 18 856967552
 20 27230880768
 22 680417833472
 24 13721772977024
 26 226128254847488
 28 3081454360189952
 30 35064826913355520
 32 336014520825141340
 34 2731238665152128768
 36 1894961228051341184
 38 112834993226032103936
 40 579364846705294996864
 42 2575849616631486204416
 44 9952155728071153882112
 46 33519982404512223401600
 48 98687914666573428364840
 50 2545742962488001599228l6
 52 5765364560406191651491l4
 54 1148237129819878789497856
 56 213890548891825020657408
```

```
 58 311403468474271539381 5552
 60 424881408802053079042 2528
 62 511834440874949289841152
 64 544586270337344451782 5478

wd.128.092.012
 12 1194816
 14 45646848
 16 2751682584
 18 110071456768
 20 3484410778688
 22 8709939355008
 24 1756359917165952
 26 28944450656120832
 28 394426389988237184
 30 4488297727663171584
 32 43009842715896693084
 34 349598717578587531264
 36 2425549189872597678976
 38 14442886028067639783424
 40 741586653206041558041 6
 42 3297089066350487847690 24
 44 12738753386272545590976
 46 42905597780091321977640 96
 48 12632047099619818751639976
 50 32585525337307036591291392
 52 73796631469243277615111 04
 54 14697442214886651424308 4288
 56 25777786830680023693247232
 58 39859662823272583189523712
 60 54384794596123339347265 4592
 62 65514839326807565887223 8080
 64 69770096246413149145713094

wd.128.099.010
 10 796544
 12 90180160
 14 6463889536
 16 347764539928
 18 14127559573120
 20 44575475469248
 22 11149685265467776
 24 224811690627712384
 26 370489537782191104
 28 50486556173121673600
 30 574502176730571255552
 32 5505259786944679990620
 34 44748635720273383143168
 36 3104702962799943929 7536
 38 18486894173013492478999 04
 40 94923091237319118515669 76
```

```
 42 42202740212894624045103744
 44 16356041742389991882232512
 46 54919165360291990896148 4160
 48 16169020228032633502641 49928
 50 41709472585828650199604 80640
 52 94459687920413917959509 26784
 54 18812726104659846681453 12896
 56 32995567020535162782224 34304
 58 51020368602278287044715 99232
 60 69612536825819432117261 21216
 62 83858994648317780352552315392
 64 89224971989631194512677986758

wd.128.106.008
  8 774192
 10 105598976
 12 11361676032
 14 828626841600
 16 44515013174520
 18 1808265733435392
 20 57056968214853376
 22 1427159096213901312
 24 28775892186952836240
 26 474226642406696116224
 28 646227907173511041 8944
 30 735362788164337729 29024
 32 704673252880779235687452
 34 57278253704580994610 38080
 36 39740197928028063063904768
 38 23663224541420383808194 9696
 40 12150155678013131756971 52304
 42 54019507474336274569812 66432
 44 20871173342366872859566014720
 46 70296531663247684816378728448
 48 20696345891289102627719 8168776
 50 53388124911384079711522 3461888
 52 12090840053465919059416 83436800
 54 24080289414648567100618 55682560
 56 42234325785062185551285 58121488
 58 65306071812806596550178 51666432
 60 89104047134463252502559 43109632
 62 10733951315294301174491841282048
 64 11420796414343588424136158689350

wd.128.113.006
  6 341376
  8 87288624
 10 13842455424
 12 1448180487936
 14 106141978256640
 16 5697211389035256
```

```
 18 231462916338818304
 20 7303265469631124224
 23 182676478544670888576
 24 3683313800412335283600
 26 60701011366229993420928
 28 8271717185445875657630 72
 30 9412643693444880033139200
 32 90198176361260636112668700
 34 7331616474282651537211008 00
 36 5086745334774298795535505920
 38 3028892741304486246612513 7280
 40 1555219926784991759059415 07120
 42 691449695671693087458634462080
 44 2671510187822394963671294035200
 46 8997956052897506127123622265600
 48 2649132274084454855472046144 0200
 50 6833679988658651159231793719 5776
 52 1547627526843289352309216571 51744
 54 3082277045075720326820005075 10400
 56 5405993700486723632424736843 64048
 58 8359177192041145268889081549 32352
 60 1140531803320872201314876100099072
 62 1373945768357978846215074177297408
 64 1461861941035652013200273232486470

wd.128.120.004
  4 85344
  6 42330624
  8 11170182384
 10 1772228014592
 12 185359804775712
 14 13586256544975872
 16 729242357526446712
 18 29627257927486958592
 20 934817955092922629344
 22 23382589365749366429184
 24 4714641660340593021227 04
 26 7769729456174562056216064
 28 1058779799704768692753855 04
 30 1204818392766796825789470720
 32 1154536657423705241877721 7820
 34 9384469087079854005243436 0320
 36 6511034028512200825869315 17920
 38 3876982708869397190103809681920
 40 1990681506284869946214005 8602480
 42 8850556104597527515220031 4606080
 44 3419533040412683458478468290 61280
 46 1151738374770880217441839661716480
 48 3390889310828097487679807613566280
 50 8747110385483091255323050018747392
 52 1980963234359406110564038479057 9552
```

```
 54 39453146176969302060067713110615552
 56 69196719366229927819863672056678992
 58 10699746805812685444195630142066272
 60 14598807082507138963311965426639721 6
 62 17586505834982158571541139235791257 6
 64 18711832845256314920999104434444960 0

wd.128.127.002
  2 8128
  4 10668000
  6 5423611200
  8 1429702652400
 10 226846154180800
 12 23726045489546400
 14 1739040916651367936
 16 93343021201262198784
 18 3792289018215984005120
 20 119656698232656988471296
 22 29929714389103547934310 40
 24 60347413251942500075044864
 26 99452537039201205626496614 4
 28 13552381436214964486037045248
 30 15421675427417015798741755494 4
 32 14778069215022799216777342484 48
 34 12012120431462387730048509542400
 36 83341235564955678220181980577792
 38 49625378673528400858712792629248 0
 40 25480723280446307864089382470287 36
 42 11328711813884834832046711017308160
 44 43770022917282358845017689455329280
 46 14742251197067275084348529683359334 4
 48 43403383178599676348799425251272294 4
 50 11196301293418358949351014497936998 40
 52 25356329399800397417247908506453934 08
 54 50500027106520717173298223989863219 20
 56 88571800788774325005710521478645022 72
 58 13695675911440237013532474696584396800
 60 18686473065609143965712255676040871936
 62 22510727468777167143671172081479843840
 64 23951146041928103937688710428982509568
```

Index

2^s-ary weight of an integer, 46

Algebraic geometry codes, 61
A-posteriori probability (APP), 122, 135
ARQ, 99
Array code, 113
Asymptotic coding gain, 173, 187
  greater than, with block partitioning, 187
Augmented code, 104
Automatic repeat request, 99
AWGN channel, 15, 122
  metric
    equivalence to changing sign, 122
  reliability, 129

BCH Bound, 45
BCH code, 44
  extended, 57
  general decoder, 48
  how to compute weight distribution, 57
  performance AWGN channel, 57
  specified by zeros, 45
BCH decoder
  Berlekamp-Massey algorithm, 47
  Euclidean algorithm, 47
  general, 48
  Massey algorithm, 63
  PGZ algorithm, 47
BCJR algorithm, 137
BEC channel, 55
Belief propagation decoding, 162
Berlekamp-Massey algorithm, 47, 49–52
  discrepancy, 49
  errors-and-erasures
    errata evaluator, 68
    errata locator, 68
    modified discrepancy, 68
    modified Forney algorithm, 68
    modified Forney syndrome, 67
Bit error probability
  AWGN, 2
Block code concept, 3
Boolean function, 27
Bound
  BCH, 45
  bit error probability, AWGN channel, 57
  Chernoff, 19
  Hamming, 11
    nonbinary case, 12
  RS decoder, 72
  union
    AWGN channel, 16
    convolutional code, 84
    multilevel modulation code for UEP, 186
Burst error correcting capability, 113

Chase algorithm, 129
  soft output, 156
    correction factor, 157
    scaling factor, 157
Chien search, 55
Code
  self-dual, 7, 29
Coded modulation
  bit-interleaved (BICM), 189
  main idea, 2
  MCM, 2, 173, 180
    multistage decoding, 182
    parallel decoding, 185
    unequal error protection, 183
  Pragmatic TCM
    symbol transformation, 177
    two-stage decoding, 176
  TCM, 2, 173, 174
    example modified state diagram, 176
    MLD decoding, 175
  turbo trellis(TTCM), 192
Coding gain, 2, 16
  asymptotic, 173, 187
Complementary error function (erfc), 2
Concatenated code, 115
Concatenated coding, 2
Conjugate elements, 43
Constellation, 170
Construction X3, 109
Convolutional code, 2
  block code obtained from, 79
  complete weight enumerator sequence, 83
  constraint length, 74

defined, 3
direct-truncation, 80
finite-state machine, 73
generator sequences, 75
polynomial generator matrix, 77
recursive systematic (RSC code), 78
RSC code, 144
state diagram, 74
tail-biting, 80
union bound over BSC and AWGN channels, 84
weight distribution block codes from, 81–84
weight enumerating sequence, 82
zero-tail code, 79
Correlation discrepancy, 133
Coset, 104
decomposition, 104, 109, 118
leader, 10
representative, 118, 140
CRC code, 37
popular polynomials, 38
Cycle set, 43
Cyclic code
defined, 33
encoding by division by $\bar{g}(x)$, 36
extended, 57
general decoder, 39
MLS code, 37
RM code, 29
shortened, 37
syndrome decoding, error-trapping, 39
zeros of, 34
Cyclic shift, 33
Cyclotomic coset, 43

Decoding
BCH codes general, 48
BCJR algorithm, 137
Belief propagation, 162
Berlekamp-Massey algorithm, 47, 49–52
Chase algorithm, 129
soft output, 156
Chien search, 55
Depth, 88
Euclidean algorithm, 47, 53–54
Forney algorithm for RS codes, 62
GMD algorithm, 132
Log-MAP algorithm, 139
look-up table, 10
MAP algorithm, 137
Massey algorithm, 63
Max-Log-MAP algorithm, 140
MLD, 86, 175
Modified Forney algorithm, 68
ordered statistics algorithm, 131
soft-output, 140
Parallel for multilevel codes, 185
PGZ algorithm, 47, 52–53
SOVA algorithm, 134
Sudan algorithm, 67, 134
SW-SOVA algorithm, 136
two-stage, 115, 176
Viterbi algorithm, 85–94
with standard array, 8
Decomposable code, 107, 117
Direct-truncation code, 80
Discrepancy Berlelamp-Massey algorithm, 49
Disjunctive normal form, 27
Distance
designed of BCH code, 44
free, 79
Hamming, 4
minimum Hamming, 4
minimum squared Euclidean, 173
squared Euclidean, 15
Dual code, 6
example, 7
of cyclic code, 37
of RM code, 28

Encoding
non-systematic, 35
recursive with parity-check matrix, 36
systematic, 16, 35
with generator matrix, 8
with parity-check matrix, 8
Erasure, 55
value, 67
Erasure correction for binary linear codes, 55
Erasure locator polynomial, 67
Error
positions, 47
values, 47
Error bursts, 2
Error correcting capability, 5
Error correcting code
as subset, 4
defined, 3
minimum Hamming distance, 4
Error evaluator polynomial, 62
Error locator polynomial, 47
Error polynomial, 46
Error propagation, 186
Euclidean algorithm, 47, 53–54
polynomials same up to a constant, 66
Euclidean geometry (EG) code, 45

Extended code, 103

Factoring polynomials in $GF(2^m)$ is hard, 55
Field, 40
  Galois, 40
    arithmetic, 41
    element order, 43
    representations, 41
Finding factors of $x^{2^m-1}+1$, 43
Finite geometry, 28
Flat Rayleigh fading channel
  bound, 18
  model, 17
Forney algorithm, 62
Fourier transform
  BCH decoding with, 67
Free distance, 79

Gallager code, 160
Galois field, 40
Generalized concatenated code, 117
  array codes, 113
Generator matrix, 6
Generator polynomial, 34
  of BCH code, 44
  of RS code, 62
GMD decoding, 132
Golay code, 25
Greatest common divisor (GCD), 53

Hamming code, 23
  shortened (71,64,3) code, 37
Hamming space
  defined, 4
  distance, 4
  sphere, 5
Hard-decision decoding, 16
  general structure, 20

Incidence vector, 29
Inner code, 110
Interleaver, 152
  block, 111
  convolutional, 116
  cyclic, 114
  Ramsey, 111, 116, 152
  random, 152
  S-random, 152
Irreducible factors, 34
Irregular LDPC code
  record performance, 3
Iterative belief propagation algorithm
  message passing, 163
Iterative belief propagation decoding, 162
Iterative bit-flip decoding, 161
Iterative decoding convergence, 152

Key equation, 47

LDPC code, 159
  error detection capability, 166
Likelihood, 85
Linear code
  as vector subspace, 6
Linear feedback shift-register (LFSR), 49
List decoding, 134
Log and antilog tables, 42
Log-likelihood metric, 122
Log-likelihood ratio (LLR), 145
  extrinsic, 146
Log-MAP algorithm, 139
Low-density parity-check code, 159

MacWilliams identity, 56
MAP algorithm, 137
Massey algorithm, 63
Matrix
  generator and parity-check, 6
  Vandermonde, 45
Max-Log-MAP algorithm, 140
Maximum-a-posteriori probability, 137
Maximum-distance-separable (MDS) code, 62
  weight distribution, 71
Maximum-length sequence (MLS) code, 37
Meggit decoder, 39
Message passing, 163
Metric
  log-likelihood, 122
Metric normalization, 90
Minimal polynomial, 43
Minimum Hamming distance, 4
Minimum squared Euclidean distance, 173
MLD decoding, 175
  defined, 15
  Viterbi algorithm, 86
Modified Forney syndrome, 67
Modified syndrome polynomial, 67
Modulation as mapping, 170
Monte Carlo integration, 19
MSED, 173
Multilevel coding, 180
Multilevel modulation code
  definition, 181
Multistage decoding, 182

Natural labeling, 174
Non-primitive BCH codes, 38
Non-systematic cyclic code, 35

Nyquist bandwidth, 169

Order of an element in $GF(2^m)$, 43
Ordered statistics decoding, 131
Orthogonal checks, 30
Outer code, 110

Parallel concatenated code, 147
Parallel decoding of multilevel codes, 185
Parity-check matrix, 6
  of BCH code, 45
  of cyclic code, 36
Parity node, 159
Parity sub-matrix, 7
Partition level, 117–118
Path memory, 87, 88
Pearl's algorithm, 162
Perfect code
  definition, 11
Permutation, 111
Permutation matrix, 147
PGZ algorithm, 47, 52–53
Polynomial
  associated with vector, 33
  erasure locator, 67
  errata locator, 68
  error, 46
  error evaluator, 62
  error locator, 47
  generator, 34
  minimal, 43
  modified syndrome, 67
  parity-check, 36
  primitive, 41
  syndrome, 39
Polynomial code, 27, 45
Primitive
  element, 41
  polynomial, 41
Probability
  a-posteriori, 135
  AWGN, Q-function, 2
  bit error, 16
  bit error, BPSK over AWGN, 2
  correct decoding, 14
  incorrect decoding, 14
  undetected error, 13
Product code, 109
  decoding, 115
Projective geometry (PG) code, 45
Punctured code, 103
Punctured convolutional codes, 94
Puncturing
  as shortening the dual code, 103

Q-function, 2

RCPC codes, 98
Reed-Muller (RM) code, 27, 45, 119
  decoder for cyclic code, 32
  majority-logic decoding, 31
  number of minimum weight codewords, 29
Reed-Solomon (RS) code, 2
  as polynomial code, 61
  binary image of, 62
  encoding as a polynomial evaluation, 61
  generator polynomial, 62
  weight distribution, 71
Reliability AWGN channel, 129
Repeat-and-accumulate code, 154
Repetition code
  example, 4, 5, 11
  probability decoding error, 14
RS decoder
  bound bit error probability, 72
  bound word error probability, 72
  error evaluator polynomial, 62
  errors-and-erasures, 67
    direct solution, 71
    errata evaluator, 68
    errata locator, 68
    modified discrepancy, 68
    modified Forney algorithm, 68
    modified Forney syndrome, 67
  Forney algorithm, 62
  Massey algorithm, 63

Self-dual code, 103
Set partitioning, 174
  block for unequal error protection, 185
  hybrid, 183
Shannon limit, 73
Shortened code, 101
  additional correctable error patterns, 102
Shortening depth, 38
Signal point, 170
Sliding window SOVA algorithm, 136
Soft decision decoding, 15
Soft-output Chase algorithm, 156
Soft-output ordered statistics algorithm, 140
SOVA algorithm, 134
Spectral efficiency, 169
Squaring construction, 108
Standard array
  as look-up table, 10
  construction, 9
  decoding, 8
State diagram
  convolutional code, 74

for computing weight distribution, 81
for computing weight enumerating sequence, 82, 176
Subcode property, 119
Sudan algorithm, 134
Sum-product algorithm, 162
Supercode, 104
Syndrome
as evaluation of zeros of code, 47
as vector, 9
circuit for computing, 48
Syndrome polynomial, 39
Syndrome trellis, 128
Systematic cyclic code, 35
Systematic encoding, 3

Tail-biting code, 80
Tanner graph, 159
Time-sharing code, 106
Trellis diagram, 76
Trellis structure
example 3-level coded 8-PSK modulation, 181
of array codes, 113
of block and convolutional codes, 3
Ungerboeck mapping, 174
Turbo code, 106, 143, 147
as a punctured product code, 148
component RSC code, 144
Two-dimensional code, 110
Two-stage decoding, 115, 176
Two-step majority-logic decoding, 31

Unequal error protection
multilevel modulation code, 183
Unequal error protection code, 99, 109, 118
example, 10

Vandermonde matrix, 45
Variable node, 159
Viterbi algorithm, 86–94
ACS, 94
branch synchronization, 89
traceback, 92
traceback memory, 76
Viterbi decoder
off-the-shelf, 177

Weight distribution
convolutional codes, 81–84
defined, 12
extended BCH codes, 57
Weight enumerating sequence, 82

Zero-tail code, 79